REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE 6/05	3. REPORT TYPE AND DATES COVERED Final. 01 June 04 - 31 May 05

4. TITLE AND SUBTITLE
Optimizing Interaction Potentials for Multi-Agent Surveillance

5. FUNDING NUMBERS
W911NF-04-1-0167

6. AUTHOR(S)
William Spears and Diana Spears

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Computer Science Department, Dept. 3315
University of Wyoming
1000 E. University Avenue, Laramie, WY 82071

8. PERFORMING ORGANIZATION REPORT NUMBER
UW-CS-AP-WMS-ARO-1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U. S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

47061.1-MA-DRP

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12 a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12 b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We have developed a physics-based control framework that provides a practical yet principled approach for designing collective systems. This framework is called "artificial physics" (AP), because agents perform actions based on virtual forces exerted on them by other agents and the environment. These forces are designed to ensure that the global behavior of a multi-agent system arises from local interactions of the agents, as well as from task-specific goals and constraints. We extend AP by using genetic algorithms (GAs) to search a space of interaction potentials so that the desired behavior emerges from the interactions between the agents. This extended framework is applied to the task of surveillance, where a team of unmanned air vehicles (UAVs) must provide maximum sensory coverage of terrain, in order to maximize the probability of detection of targets of interest. This report summarizes preliminary results that indicate that robust behavior is achieved, despite loss of assets or sensor degradation. This report also provides some initial theoretical analyses of simple behavior-based asset controllers on the surveillance task.

14. SUBJECT TERMS
surveillance, artificial physics, interaction potentials, genetic algorithms, simulation testbed

15. NUMBER OF PAGES
103

16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION ON THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	**UNCLASSIFIED**	**UNCLASSIFIED**	**UL**

Optimizing Interaction Potentials for Multi-Agent Surveillance: Final Report

William M. Spears
Diana F. Spears
Wesley Kerr
Suranga Hettiarachchi
Dimitri Zarzhitsky

Computer Science Department,
University of Wyoming, Laramie, WY, 82071, USA
wspears@cs.uwyo.edu
http://www.cs.uwyo.edu/∼wspears

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Forward

We have developed a physics-based control framework that provides a practical yet principled approach for designing collective systems. This framework is called "artificial physics" (AP), because agents perform actions based on virtual forces exerted on them by other agents and the environment. These forces are designed to ensure that the global behavior of a multi-agent system arises from local interactions of the agents, as well as from task-specific goals and constraints. We extend AP by using genetic algorithms (GAs) to search a space of interaction potentials so that the desired behavior emerges from the interactions between the agents. This extended framework is applied to the task of surveillance, where a team of unmanned air vehicles (UAVs) must provide maximum sensory coverage of terrain, in order to maximize the probability of detection of targets of interest. This report summarizes preliminary results that indicate that robust behavior is achieved, despite loss of assets or sensor degradation. This report also provides some initial theoretical analyses of simple behavior-based asset controllers on the surveillance task.

Part I

Report

Chapter 1

Introduction

1.1 The Task

The task to be addressed is multi-agent surveillance. Assume there are α assets flying at a constant altitude over terrain that contains areas of foliage and non-foliage. The assets have sensors to detect targets of interest (such as tanks), and they can determine whether they are over areas of non-foliage or foliage. The target detectors cannot penetrate the foliage – thus, regions of foliage are of lower interest than those without foliage. Each asset has a target detector with a field of view of $\pi {r_t}^2$, and a terrain detector with a field of view of $\pi {r_f}^2$, where $r_f > r_t$. The total area of the region is much greater than $\alpha \pi {r_t}^2$. The goal is to locate T targets as reliably as possible, within some time period t. Targets may move and may hide in foliage.

Chapter 2

The Environment

2.1 Forest Generation

For this project we assumed that foliage is synonymous with a forest of trees. When the targets are under the trees they are undetectable, but otherwise they are seen by the UAVs. In order to simulate this problem, we created an algorithm that generates random forests, i.e. refuge for the targets. The algorithm used to generate these forests includes a stochastic element, therefore with different random number seeds, we generate different forests with roughly the same amount of coverage. But, we also needed the capability to rerun experiments generating the same forest as before. Given a certain random seed the algorithm is deterministic and repeatable. Different random seeds provide the stochastic element.

Before we describe the algorithm we need a few definitions. Let the world be a square such that the sides are of length L. This is a discrete world, therefore we can represent a forest as a two-dimensional binary table. In the world, the position (i,j) can either be a collection of trees (referred to from here on as a tree) or can remain barren. We designed the algorithm to mimic a real forest, such that there are larges clusters of trees, yet there can be holes in the middle. These holes allow assets to view targets.

The algorithm begins by placing an initial number of trees in the environment. The algorithm uses the natural process of seeding in order to generate the next phase of the forest. The process continues until a desired state has been reached. The controlling parameters of the algorithm are:

- n_f - The initial number of trees in the forest.

- R_s - The distance that seeds can travel.
- S_s - The amount of seeds spread during seeding.
- d_s - The decay rate of the seeds.
- F_c - The desired forest coverage.

The algorithm begins by initializing n_f trees. The placement of these trees is uniformly distributed about the world. Rather than just initializing one square for a clump of trees, we initialize a cross section, to aid in the development of the forest, see Fig 2.1(a). Once the initial trees have been placed, the next generation begins. For a given generation the current trees in the population are allowed to seed the areas surrounding them with a distance of R_s. The amount of seed laid at each position within this circle is defined as S_s. The seeding process can be seen in Fig 2.1(b). Once everything has been seeded, we begin selecting positions (i,j) that are seeded and determine if they become trees, by selecting a uniformly distributed random number. If the value selected is less than that of the seeds in the area, then the seeds have taken hold, and grow to become trees. Using this procedure not all seeds will become trees therefore, we apply a decay to those seeds that have not become trees. Notice that as more trees surround an area, the greater the probability of growing a tree, since it has more seeds. We continue the algorithm until the desired forest coverage has been achieved. Therefore, upon completion we don't guarantee that there is exactly F_c coverage, but at least that much coverage.

Setting the parameters is relatively easy, knowing that at full seed strength, we will have perfect circles of trees of size R_s. As we decrease S_s we begin getting more sparsely populated forests because our seeds are not as hearty and don't survive. The decay rates provide a mechanism to remove old seeds from the population of seeds, again protecting us from realizing perfect circled forests. Figure 2.2 provides a nice illustration of our forest generator.

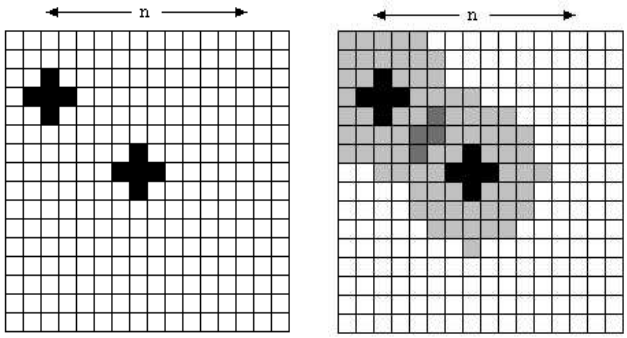


Figure 2.1: Forest seeding algorithm. *(a)* The initial forest configuration. Notice that the trees do not represent exactly one square. *(b)* The seeding portion of the algorithm. Notice that the overlap receives more seeds and is more likely to create a new tree.

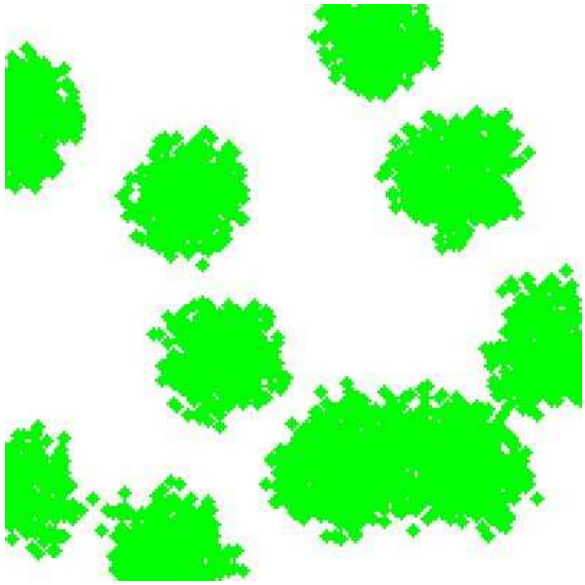


Figure 2.2: An Example Forest.

Chapter 3

Physics-Based Asset Control

3.1 Asset Control – The Artificial Physics Framework

The UAVs are assumed to have a "terrain sensor" and a "target sensor". The field of view of the target sensor is presumed to be considerably less than the field of view of the terrain sensor. The target sensor has a probability of detection P_d. The terrain sensor returns a number from 0.0 to 1.0 that represents the percentage of foliage currently below the vehicle. Our control strategy assumes that this latter information can be used by each UAV to determine its desired separation from other UAVs.

The UAVs motion is controlled via the "artificial physics" (AP) framework. Virtual physics forces drive a multi-agent system to a desired configuration or state. The desired configuration (state) is one that minimizes overall system potential energy. In essence the system acts as a molecular dynamics ($\vec{F} = m\vec{a}$) simulation.

At an abstract level, AP treats agents as physical particles. This enables the framework to be embodied in vehicles ranging in size from nanobots to satellites. Particles exist in two or three dimensions and are point-masses. Each particle i has position $\vec{x}$ and velocity $\vec{v}$. We use a discrete-time approximation to the continuous behavior of the system, with time-step Δt. At each time step, the position of each particle undergoes a perturbation $\Delta\vec{x}$. The perturbation depends on the current velocity, i.e., $\Delta\vec{x} = \vec{v}\Delta t$. The velocity of each particle at each time step also changes by $\Delta\vec{v}$. The change in velocity is controlled by the force on the particle, i.e., $\Delta\vec{v} = \vec{F}\Delta t/m$, where m is the mass of that particle and $\vec{F}$ is the force on that particle.[1] A frictional force is included, for self-stabilization. This is modeled as a

[1] F and v denote the magnitude of vectors $\vec{F}$ and $\vec{v}$.

viscous friction term, i.e., the product of a viscosity coefficient and the agent's velocity.

For the current research, two general force laws are examined. The first is a generalization of the "gravitational" force law to include both attraction and repulsion. The force law is:

$$F = \frac{Gm_i m_j}{r^p} \tag{3.1}$$

$F \leq F_{max}$ is the magnitude of the force between two UAVs i and j, and r is the distance between the two UAVs. The variable p is a user-defined power, which ranges from 0.0 to 5.0. When $p = 0.0$ the force law is constant for all distances. Currently, $m_i = 1.0$ for all UAVs. The force is repulsive if $r < R$ and attractive if $r > R$. R is the desired separation between that UAV and neighboring UAVs, and depends on the percentage of foliage currently below that UAV. In order to achieve optimal behavior, the values of G, p, and F_{max} must be determined, as well as the amount of friction and the mapping of the percentage of foliage to the desired separation R.

Our second force law is a generalization of the Lennard-Jones law, where:

$$F = 24\epsilon \left[\frac{2d(\sigma_i + \sigma_j)^{12}}{r^{13}} - \frac{c(\sigma_i + \sigma_j)^6}{r^7} \right] \tag{3.2}$$

Again, $F \leq F_{max}$ is the magnitude of the force between two UAVs i and j, and r is the distance between the two UAVs. The variable ϵ affects the strength of the force, while c and d control the relative balance between the attractive and repulsive components. In order to achieve optimal behavior, the values of ϵ, c, d, and F_{max} must be determined, as well as the amount of friction and the mapping of the percentage of foliage to the desired separation σ.

Chapter 4

Behaviors-Based Asset Control

4.1 Asset Control – Behavior-based Control

Due to the complexity in behavior of the physics-based controllers, we also implemented several behavior-based controllers. Some of these controllers are amenable to mathematical analysis (see Section 10.1).

- **Bouncy:** simply moves the assets vertically, "bouncing" off the top and bottom world boundaries by reversing the UAV velocity upon reaching the edge of the surveillance area. This is a realistic controller.

- **Column:** is a variation on plain vertical movement of the UAVs, except the world is assumed to be toroidal, so that the assets "wrap around" in the same vertical column when moving past the top and bottom edges of the surveillance area. Although this is not realistic for one asset, it could work quite well for multiple assets in adjacent columns.

- **MagicCarpetArea:** is an asset controller useful in testing statistical models of the surveillance problem, in which assets "appear" at random locations throughout the world during the simulation. UAV locations are selected using a uniform random distribution such that target sensor footprints do not overlap.

- **MagicCarpetColumn:** similar in spirit to the **MagicCarpetArea**, this controller causes assets to appear at random locations, but restricts the surveillance region to a single column, so that the x coordinate of the asset remains fixed, as the y coordinate

changes randomly, without target sensor footprint overlap between adjacent asset locations.

- **MagicCarpetDiagonal:** causes assets to appear at random locations along both of the major diagonals of the environment.

- **SL:** a realistic "Straight Line" asset controller which ignores foliage distribution, and simply moves an asset in a straight line at a constant speed until it reaches a world boundary, at which point the UAV will "reflect" from the edge by reversing its velocity component in the direction perpendicular to the boundary, similar to how a billiard ball bounces from an edge of the table.

- **SSLAF:** is a sophisticated controller ("Super Straight Line Avoid Forest") based on the premise that any surveillance time spent over foliage (which inhibits target detection) is in some sense "wasted", and thus attempts to minimize UAV flight time over forested areas. At each time step, it uses its forest sensor to compute the percentage of foliage visible through the sensor, and if that percentage exceeds a specified threshold (the default is 10%), the asset then estimates the mass distribution of the tree canopy underneath, and moves the asset directly away from the center of foliage mass. Once the detected foliage percentage drops below the threshold, the asset will switch into its normal surveillance mode, continuing to move with its present velocity until an encounter with another patch of foliage or a bounce from the surveillance perimeter takes place. Should the asset be initially deployed directly over the foliage, it will continue moving in its initial direction until it leaves the foliage, or an interaction with a world boundary causes it to turn around; thus given sufficient time, the asset will find a foliage-free region.

- **SLAF:** is a simplification of the **SSLAF** controller, motivated by the fact that the UAV foliage sensor may not be able to detect a distribution of individual trees (or groups of trees) in the foliage cover, and thus the computation of the center of mass of the tree canopy may be impossible or impractical to perform. Instead, this controller simply reverses an asset's course when the amount of foliage detected by the sensor

exceeds a certain threshold, which is currently set to 50%. In the absence of the foliage, **SLAF** and **SSLAF** are equivalent to the **SL** controller.

- **Side:** controller produces linear "sweeping bands" of target sensor coverage by moving the UAVs back and forth along a linear path between two points on world edges. The location and direction of these surveillance corridors is determined by the initial location and velocity of each UAV, and may not include areas near the world boundary, as the assets may reverse their course right after detecting the edge of the surveillance area, and before scanning the boundary region with their target sensor.

- **Stationary:** assets hover over a fixed location in the surveillance perimeter, scanning any targets that may pass underneath. This is a simple controller that does not attempt to navigate around areas hidden by the foliage.

Chapter 5

Target Control

5.1 Target Control

For our baseline experiments our targets were stationary. We also have a target controller for moving targets. The goal of the targets is to enter the environment from the left, cross the environment, and exit from the right. The idea behind this "Gollum" (i.e., sneaky) controller is to bias the targets to choose a path with maximum foliage, so that the time a target spends exposed to the assets' sensors is minimized. To achieve the effect, each target has two attributes, a foliage sensor radius, and a foliage sensor angle, which define how far and how wide the foliage sensor scans the environment ahead of the target to find foliage. Each target attempts to move from left to right, and in the absence of foliage, it will move at 0 degrees to the horizontal to maximize its speed toward the right edge. However, when foliage is detected (which is a simple first hit-or-miss detection, no analysis is performed as to how far or how large the foliage is), the target will alter its course to reach the foliage cover at maximum speed. Once in the foliage, the target will again continue moving toward the right edge, and attempt to stay within the foliage as long as possible.

For now, the target foliage sensor radius is 10 (in a 200×200 world), and the foliage sensor angle is 60 degrees, so that the sensor scans +30 degrees and -30 degrees away from the horizontal. These values seem to experimentally give a decent trade-off between seeking foliage cover and avoiding large detours in target paths.

We have also used an alternative "Gollum" (i.e., avoid) controller, predicated on the assumption that the UAV sensor can leave a footprint that is detectable by the target. If the footprint is detected, then the target will pause for 3 time steps. If the signal is still

there, the target will attempt to go around the sensor (by making at most a 90 degree turn either up or down, until it no longer detects the sensor, at which point it will proceed with its forward motion).

We have implemented a suite of target controllers, summarized as follows:

- **Avoid:** targets attempt to actively avoid the ground-level sensor footprint of the overhead UAVs while crossing the area under surveillance. When a moving target detects a UAV sensor beam in its path, it pauses for a short time, waiting for the UAV to move away; if the sensor beam persists after the delay, then the target will attempt to circumnavigate around the sensor footprint.

- **Ribbon:** targets follow a periodically sinuous trajectory, which varies slightly for each target. The amplitude and period of the mean-path oscillation are kept low to approximate realistic deviations from straight line travel due to terrain configuration.

- **Sneaky:** targets aim to minimize their exposure time to the UAV sensors by attempting to maximize the amount of travel time spent underneath the foliage. This target controller assumes that the targets may scan their local environment for foliage-covered areas, and will select the next waypoint to be inside the forest if possible.

- **Stationary:** targets simply do not move.

- **StraightLine:** targets move in a horizontal line formation from the left world boundary to the right edge of the surveillance area.

- **MilitaryCombo:** targets move in a combination of wedge, vee, echelon left/right, and column formations from the left world boundary to the right boundary of the surveillance perimeter. These target groupings mimic the actual tank battlefield formations, and a representative sample of each formation type is shown in Fig. 5.1. These formations are modeled after real-life scenarios, as found in the Department of the Army's Field Manual "Tank and Mechanized Infantry Company Team" (FM-71-1, 2002).

Figure 5.2 shows areas of forest, and 100 tanks. The triangle represents a tank that has not yet been seen but is visible, and the "×" represents a hidden tank that has not been

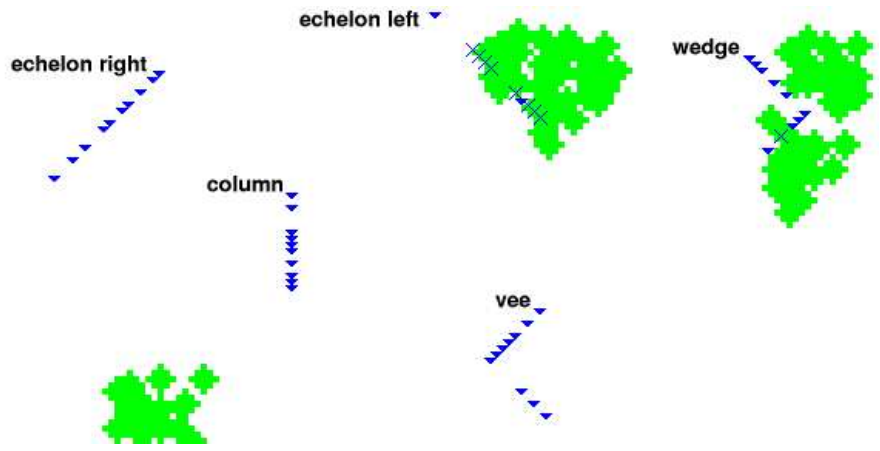


Figure 5.1: Targets arranged in the wedge, vee, echelon left/right, and column formations.

seen. The simulation also uses a "+" to represent a tank that has been seen and is visible, and a "|" to represent a tank that is currently hidden but has been previously seen.

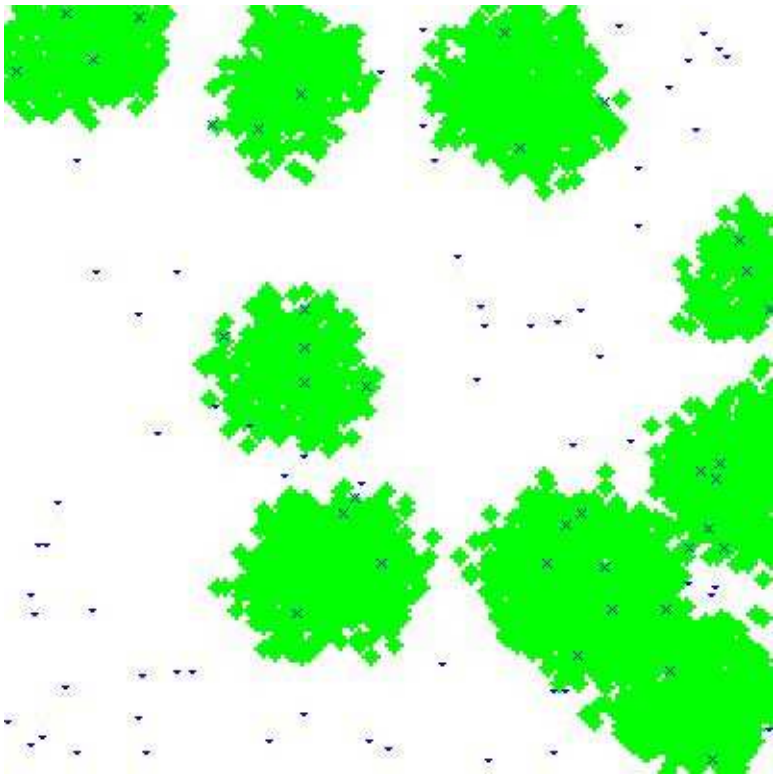


Figure 5.2: The surveillance environment.

Chapter 6

Genetic Algorithms

6.1 Optimizing Parameters Using Genetic Algorithms

Given a generalized asset interaction potential, such as the standard AP force law or Lennard-Jones (LJ), it is necessary to optimize the parameters to achieve the best performance. We achieve this task using a genetic algorithm (GA). GAs are optimization algorithms inspired by natural evolution. We mutate and recombine a population of candidate solutions (individuals) based on their performance (fitness) in our environment. The individuals that have higher fitness than the average fitness of the population will reproduce and contribute their genetic makeup to future generations. One of the major reasons for using this population-based stochastic algorithm is that it quickly generates individuals that have robust performance.

We use a population size of 100 individuals and termination after 150 generations. The genetic algorithm randomly initializes the first population, which is controlled via a random number seed. Every individual in the population represents one possible instantiation of either the AP or LJ potential, based on the potential function being optimized.

Further discussion of the genetic algorithm needs clear definitions of the parameter sets used in these potential functions. Table 6.1 shows the individual representation of parameters for the standard AP potential and Table 6.2 shows the representation of parameters for the Lennard-Jones potential.

The evolving parameters of the Artificial Physics potential are:

- G_a - gravitational constant of asset-asset interactions.
- F_{max_a} - maximum force of asset-asset interactions.

Table 6.1: Artificial Physics Individual

G_a	F_{max_a}	p_a	G_w	F_{max_w}	p_w	c_a	c_b	c_c	Fr

- p_a - power of the force law of asset-asset interactions.
- G_w - gravitational constant of wall-asset interactions.
- F_{max_w} - maximum force of wall-asset interactions.
- p_w - power of the force law of wall-asset interactions.
- c_a - coefficient in $c_a x^2 + c_b x + c_c$ which determines asset radii.
- c_b - coefficient in $c_a x^2 + c_b x + c_c$ which determines asset radii.
- c_c - coefficient in $c_a x^2 + c_b x + c_c$ which determines asset radii.
- Fr - friction in the system.

Table 6.2: Lennard-Jones Individual

ϵ	F_{max_a}	c	d	G_w	F_{max_w}	p_w	c_a	c_b	c_c	Fr

These evolving parameters of the Lennard-Jones potential are:

- ϵ - depth of the potential well.
- F_{max_a} - maximum force of asset-asset interactions.
- c - non-negative attractive component.
- d - non-negative repulsive component .
- G_w - gravitational force of wall-asset interactions.
- F_{max_w} - maximum force of wall-asset interactions.

- p_w - power of the force law of wall-asset interactions.
- c_a - coefficient in $c_a x^2 + c_b x + c_c$ which determines asset radii.
- c_b - coefficient in $c_a x^2 + c_b x + c_c$ which determines asset radii.
- c_c - coefficient in $c_a x^2 + c_b x + c_c$ which determines asset radii.
- Fr - friction in the system.

The performance of each individual is the percentage of targets detected by the agent collective within some period of time.

$$fitness = \frac{targets\text{-}seen}{total\text{-}number\text{-}of\text{-}visible\text{-}targets} * 100.0 \tag{6.1}$$

The selection algorithm used by the parents to generate offspring is Baker's Stochastic Universal Sampling algorithm and offspring are generated using one-point crossover with a crossover rate of 60%, and two different mutation operators. The first mutation operator modifies a parameter by adding/subtracting an amount drawn from the interval [0, δ] with uniform probability. The second mutation operator adds/subtracts an amount drawn from a Gaussian distribution with 0.0 mean and δ standard deviation. Each parameter has a $1/p$ probability of being mutated (where p is the number of parameters in an individual). Both of these operators ensure that the values in the parameter set stay within the accepted range of values. Tables 6.3 and 6.4 below show the range of values for different parameters for the Artificial Physics and Lennard-Jones potentials.

Table 6.3: Ranges of Values for Artificial Physics Individual

	G_a	F_{max_a}	p_a	G_w	F_{max_w}	p_w	c_a	c_b	c_c	Fr
max	5000.0	5.0	2.0	5000.0	5.0	2.0	22.0	22.0	22.0	1.0
min	100.0	1.0	0.1	100.0	1.0	0.1	5.0	5.0	5.0	0.0

Table 6.4: Ranges of Values for Lennard-Jones Individual

	ϵ	F_{max_a}	c	d	G_w	F_{max_w}	p_w	c_a	c_b	c_c	Fr
max	50.0	5.0	20.0	20.0	5000.0	5.0	2.0	22.0	22.0	22.0	1.0
min	1.0	1.0	1.0	1.0	100.0	1.0	0.1	5.0	5.0	5.0	0.0

Chapter 7

Stationary Target Results

7.1 Results with Stationary Targets

For our baseline studies the targets were stationary. The 200×200 environment contains 20% foliage, 100 targets (some of which may not be visible) and 10 assets. The simulation ran for 1000 time steps, assessing the quality of each individual within the GA. After the GA is run, the best individual ever seen is extracted for further testing. Several experiments were performed with that individual. The first was to generate 10 other environments with the same characteristics. The second was to vary the percentage of foliage. The third examined the performance when more assets are added, or assets are lost. Finally, the fourth experiment examined the performance when the probability of detection P_d is decreased.

7.1.1 AP Results

In this subsection we examine the best individual found by the GA, when operating with the standard AP force law. Table 7.1 gives the performance of the best individual on 10 different environments with 20% foliage, 100 targets, and 10 assets. Results are averaged over 10 independent runs. Table 7.2 shows the performance of the best individual on environments where the percentage of foliage changes. As can be seen, the AP individual achieves better than 99% performance ranking in all circumstances.

The behavior exhibited by the best AP individual is interesting in that the assets act as if they were connected by springs. The emergent behavior is a pulsation of assets, providing robust coverage of the environment. This behavior is robust with respect to the addition or deletion of assets and the lowering of the detection probability for the target sensor (see

Table 7.1: Percentage of Targets Found by AP Potential

Env	AP
1	100.000000
2	99.090909
3	99.508197
4	100.000000
5	99.032258
6	99.850746
7	100.000000
8	99.852941
9	99.824561
10	99.538462

Table 7.2: Percentage of Targets Found by AP Potential

Foliage (%)	AP
0	100.000000
10	100.000000
20	99.750000
30	99.180328
40	99.818182
50	99.545455
60	100.000000
70	100.000000
80	100.000000
90	100.000000

Figure 7.1 for 20% foliage).

7.1.2 LJ Results

In this subsection we examine the best individual found by the GA, when operating with the Lennard-Jones force law. Table 7.3 shows the performance of the best individual on 10 different environments with 20% foliage, 100 targets, and 10 assets. Results are averaged over 10 independent runs. Table 7.4 shows the performance of the best individual on environments where the percentage of foliage changes. As can be seen, the LJ individual

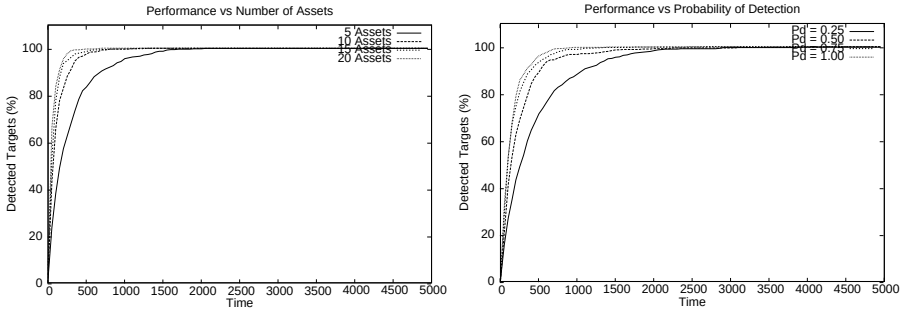


Figure 7.1: Performance of the optimal AP individual.

achieves an even better performance than the AP individual.

Table 7.3: Percentage of Targets Found by LJ Potential

Env	LJ
1	100.000000
2	100.000000
3	100.000000
4	100.000000
5	100.000000
6	99.846154
7	100.000000
8	100.000000
9	100.000000
10	100.000000

The behavior exhibited by the best LJ individual is interesting in that the assets act as if they were gas molecules. This appears to be a consequence of the very short interaction range. The emergent behavior again provides robust coverage of the environment. This behavior is robust with respect to the addition or deletion of assets, and the lowering of the detection probability for the target sensor (see Figure 7.2).

Figure 7.3 shows the Lennard-Jones potential operating with three assets. Notice that there are three circles surrounding each asset. The innermost circle represents the target

Table 7.4: Percentage of Targets Found by LJ Potential

Foliage (%)	LJ
0	100.000000
10	100.000000
20	100.000000
30	100.000000
40	99.830508
50	100.000000
60	99.756098
70	100.000000
80	100.000000
90	100.000000

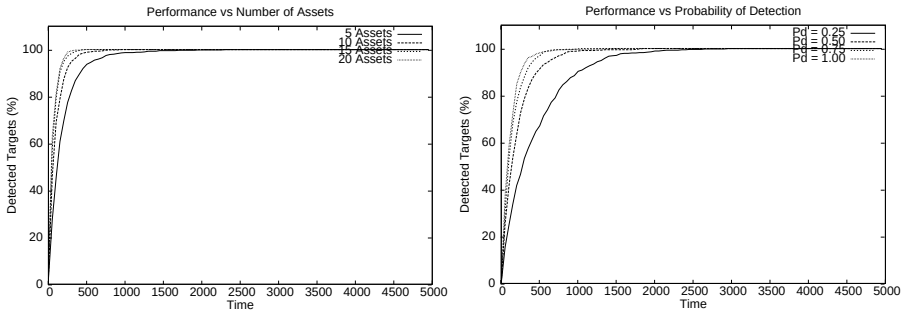


Figure 7.2: Performance of the optimal LJ individual.

sensor field of view. For each of the assets another circle, which has a larger radius, represents the field of view of the foliage sensor. The third circle, which represents the desired separation of the asset from other assets, is different in size for each agent. The smallest circle is around asset '1', since it is not over any foliage. The next smallest is around asset '2', since it is partially over foliage. The largest is around asset '0', since it is almost entirely over foliage. For stationary targets this strategy makes sense, since there is no point in clustering assets over foliage.

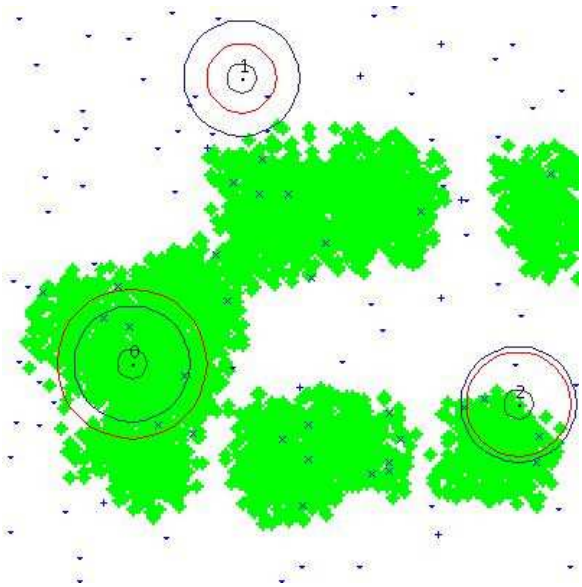


Figure 7.3: The surveillance environment,

Chapter 8

Sneaky Target Results

8.1 Results with Moving "Sneaky" Gollum Targets

We next allowed the targets to be controlled via their "Sneaky" controller. The experimental methodology is the same as above.

8.1.1 AP Results

In this subsection we examine the best individual found by the GA, when operating with the standard AP force law. Table 8.1 compares the performance of the best individual on 10 different environments with 20% foliage, 100 targets, and 10 assets. Results are averaged over 10 independent runs. Table 8.2 compares the performance of the best individual on environments where the percentage of foliage changes. The performance of the AP individual has decreased. Although still quite good for lower percentage of foliage, the AP individual starts to degrade with high percentage of foliage. In these situations the "Gollum" controller ensures that the targets spend most of their time under cover, and assets spend too much time flying over foliage.

Figure 8.1 illustrates the performance (over 20% foliage) with respect to the addition or deletion of assets and the lowering of the detection probability for the target sensor. Clearly, removing assets is more problematic in this situation, because halving the number of assets performs worse than halving the probability of detection.

Table 8.1: Percentage of Targets Found by AP Potential

Env	AP
1	90.800000
2	91.800000
3	95.700000
4	94.949495
5	93.800000
6	91.900000
7	88.600000
8	84.300000
9	85.800000
10	91.600000

Table 8.2: Percentage of Targets Found by AP Potential

Foliage (%)	AP
0	98.200000
10	98.400000
20	96.000000
30	95.757576
40	82.300000
50	42.727273
60	72.375000
70	53.797468
80	2.584270
90	30.000000

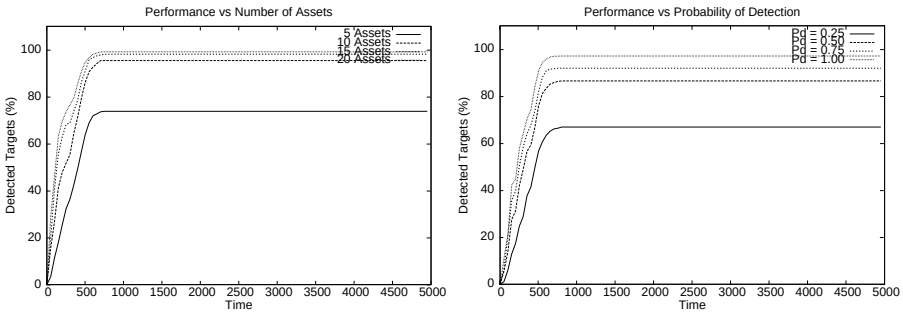


Figure 8.1: Performance of the optimal AP individual.

8.1.2 LJ Results

In this subsection we examine the best individual found by the GA, when operating with the Lennard-Jones force law. Table 8.3 compares the performance of the best individual in 10 different environments with 20% foliage, 100 targets, and 10 assets. Results are averaged over 10 independent runs. Table 8.4 compares the performance of the best individual in environments where the percentage of foliage changes. As can be seen, the LJ individual again achieves better performance than the AP individual, although performance still degrades somewhat with a higher percentage of foliage.

Table 8.3: Percentage of Targets Found by LJ Potential

Env	LJ
1	96.700000
2	91.900000
3	96.400000
4	88.200000
5	90.700000
6	95.800000
7	93.000000
8	89.800000
9	84.200000
10	91.700000

Figure 8.2 illustrates the performance (over 20% foliage) with respect to the addition or deletion of assets and the lowering of the detection probability for the target sensor. Lowering the detection probability is more problematic in this situation.

Table 8.4: Percentage of Targets Found by LJ Potential

Foliage (%)	LJ
0	99.400000
10	98.300000
20	95.100000
30	89.600000
40	93.131313
50	73.300000
60	58.900000
70	67.407407
80	44.810127
90	41.176471

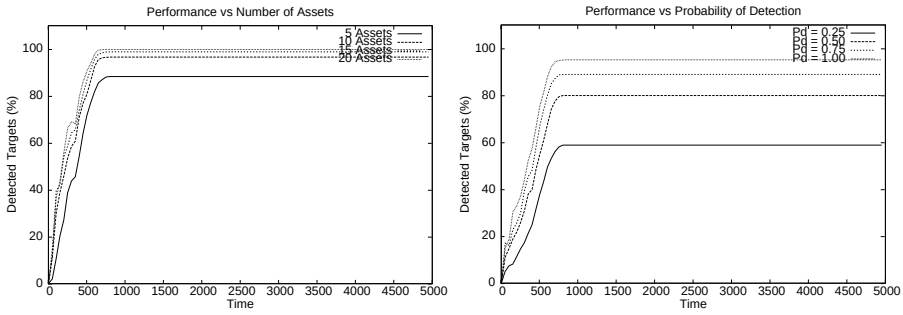


Figure 8.2: Performance of the optimal LJ individual.

Chapter 9

Avoid Target Results

9.1 Results with Moving "Avoid" Gollum Targets

We finally allowed the targets to be controlled via their "Avoid" controller. The experimental methodology is the same as above.

9.1.1 AP Results

In this subsection we examine the best individual found by the GA, when operating with the standard AP force law. Table 9.1 compares the performance of the best individual on 10 different environments with 20% foliage, 100 targets, and 10 assets. Results are averaged over 10 independent runs. Table 9.2 compares the performance of the best individual on environments where the percentage of foliage changes.

Table 9.1: Percentage of Targets Found by AP Potential

Env	AP
1	52.550000
2	44.850000
3	49.200000
4	50.550000
5	50.900000
6	51.950000
7	53.650000
8	43.000000
9	55.900000
10	51.500000

Table 9.2: Percentage of Targets Found by AP Potential

Foliage (%)	AP
0	65.750000
10	53.050000
20	60.300000
30	56.050000
40	30.600000
50	41.200000
60	26.130952
70	17.650000
80	23.417722
90	14.875000

Figure 9.1 illustrates the performance (over 20% foliage) with respect to the addition or deletion of assets and the lowering of the detection probability for the target sensor. Clearly, removing assets is more problematic in this situation, because halving the number of assets performs worse than halving the probability of detection.

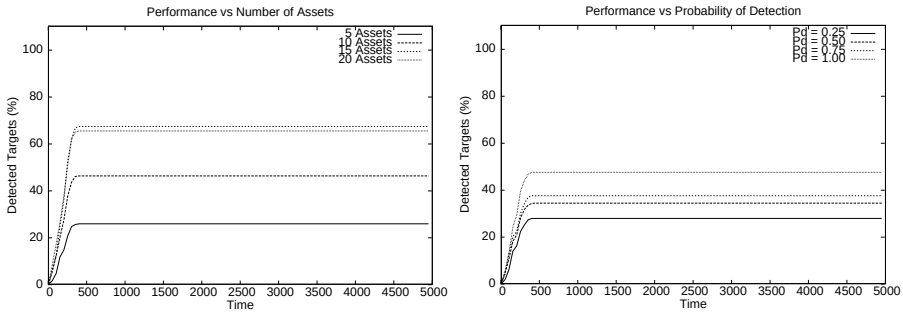


Figure 9.1: Performance of the optimal AP individual.

9.1.2 LJ Results

In this subsection we examine the best individual found by the GA, when operating with the Lennard-Jones force law. Table 9.3 compares the performance of the best individual in

10 different environments with 20% foliage, 100 targets, and 10 assets. Results are averaged over 10 independent runs. Table 8.4 compares the performance of the best individual in environments where the percentage of foliage changes. As can be seen, the LJ individual again achieves better performance than the AP individual, although performance still degrades somewhat with a higher percentage of foliage.

Table 9.3: Percentage of Targets Found by LJ Potential

Env	LJ
1	86.950000
2	88.250000
3	83.400000
4	83.050000
5	85.250000
6	82.350000
7	81.800000
8	83.900000
9	84.600000
10	86.200000

Table 9.4: Percentage of Targets Found by LJ Potential

Foliage (%)	LJ
0	88.450000
10	90.650000
20	89.150000
30	85.400000
40	76.150000
50	65.150000
60	61.700000
70	43.800000
80	36.275862
90	35.326150

Figure 9.2 illustrates the performance (over 20% foliage) with respect to the addition or deletion of assets and the lowering of the detection probability for the target sensor.

Interestingly, halving the number of assets is roughly analogous to halving the probability of detection.

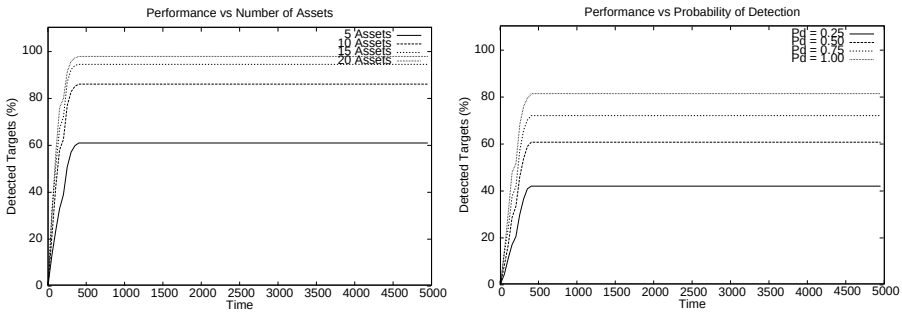


Figure 9.2: Performance of the optimal LJ individual.

Chapter 10

Analysis

10.1 Mathematical Analysis

We have also initiated a mathematical analysis of some simpler behavior-based asset controllers (see also "Strategies for Multi-Asset Surveillance", presented at a special session on cooperative agents at IEEE NSC 2005).

First, for the sake of tractability, we assume that the $L \times L$ environment contains no forest, that $P_d = 1.0$, and that T targets are crossing the environment horizontally from left to right. We consider several of the behavior-based controllers, with two types of target detector. One target detector is square (i.e., a digital camera), that covers an area $4{r_t}^2$. The other target detector is circular, covering an area of $\pi {r_t}^2$.

10.1.1 Horizontal/Side Movement

This situation is the easiest to analyze. If one asset has target radius r_t, then it will sweep a row with height $2r_t$. If T targets are uniformly distributed along the left-most column as they start their movement, then the asset will detect $2r_tT/L$ targets. If there are α assets with non-overlapping horizontal paths, then the expected number of targets detected is:

$$\frac{\alpha 2 r_t T}{L} \tag{10.1}$$

Note that the velocity of the asset is not relevant for this particular situation. The velocity of the target must be greater than zero. This result is the same regardless of the form of target sensor.

10.1.2 Magic Carpet Column

Again, assume the asset has target radius r_t. It will sweep a column of width $d = 2r_t$. Assume for the sake of simplicity that the velocity of the asset $v = 2r_t$, so that overlap does not occur. Then the number of steps required for the asset to traverse the column $S_\alpha = L/d$. Thus, assuming a square target sensor, any grid point in the column is hit with a probability approximated by $1/S_\alpha$, and is not hit with probability approximated by $1 - 1/S_\alpha$. Finally, we need to calculate how long a *target* will be within that column. If the speed of the target is v_t, then the target will require $S_t = d/v_t$ steps to cross the column. Hence the probability of that target not being detected is approximately $(1 - 1/S_\alpha)^{S_t}$. Finally, the expected number of targets detected is approximately:

$$T\left[1 - \left(1 - \frac{1}{S_\alpha}\right)^{S_t}\right] \tag{10.2}$$

If there are α independent assets then the number of targets detected is:

$$T\left[1 - \left(1 - \frac{1}{S_\alpha}\right)^{\alpha S_t}\right] \tag{10.3}$$

If the target sensor is circular, the number of targets detected is:

$$T\left[1 - \left(1 - \frac{\pi}{4S_\alpha}\right)^{\alpha S_t}\right] \tag{10.4}$$

10.1.3 Magic Carpet Diagonal

This situation is a small transformation of vertical column movement. Since moving along the diagonal is a factor of $\sqrt{2}$ longer than moving along the column, both S_α and S_t must be renormalized. Assuming a square target detector, in this situation $S_\alpha = \sqrt{2}L/d$ and $S_t = \sqrt{2}d/v_t$. If there are α independent assets then the number of targets detected is:

$$T\left[1 - \left(1 - \frac{1}{S_\alpha}\right)^{\alpha S_t}\right] \tag{10.5}$$

If the target sensor is circular, the number of targets detected is:

$$T\left[1-\left(1-\frac{\pi}{4S_\alpha}\right)^{\alpha S_t}\right] \tag{10.6}$$

10.1.4 Magic Carpet Area

Finally, if an asset is uniformly covering the domain then (assuming a square target sensor), $S_\alpha = L^2/4{r_t}^2$ and $S_t = L/v_t$. If there are α independent assets then the number of targets detected is:

$$T\left[1-\left(1-\frac{1}{S_\alpha}\right)^{\alpha S_t}\right] \tag{10.7}$$

If the target sensor is circular, $S_\alpha = L^2/\pi{r_t}^2$.

10.1.5 Straight Line

In the aggregate, a collection of assets moving with the SL controller act very much like the Magic Carpet Area controller (all portions of the environment are uniformly covered). Hence (assuming a square target sensor), $S_\alpha = L^2/4{r_t}^2$ and $S_t = L/v_t$. If there are α independent assets then the number of targets detected is:

$$T\left[1-\left(1-\frac{1}{S_\alpha}\right)^{\alpha S_t}\right] \tag{10.8}$$

If the target sensor is circular, $S_\alpha = L^2/\pi{r_t}^2$.

10.1.6 Toroidal Column

As with the Magic Carpet Column controller, $S_\alpha = L/d$ and $S_t = d/v_t$. Assuming a square target controller, then (with one asset) the number of targets detected is:

$$T, \quad \text{if } S_t > S_\alpha \tag{10.9}$$

$$\frac{TS_t}{S_\alpha}, \quad \text{if } S_t \leq S_\alpha \tag{10.10}$$

With a circular target controller, the number of targets is well estimated by:

$$T\left[1-\left(1-\frac{\pi}{4}\right)^{\frac{S_t}{S_\alpha}}\right], \quad \text{if } S_t > S_\alpha \tag{10.11}$$

$$\frac{T\pi S_t}{4S_\alpha}, \quad \text{if } S_t \leq S_\alpha \tag{10.12}$$

10.1.7 Bouncy Column

The analysis of the "bouncy" column controller is more complex. The variable S_t can be thought of as the number of time steps needed by the target to cross the column created by the asset. The idea for the bouncy theory came from the observation that at any given point in time, a set of targets enters an assets' column sweep area. The asset then has S_t time steps to detect that set of targets. When viewed this way we observe that the asset is equally likely to be in any of the S_α positions and each one allows the asset to cover a certain number of *new* sensor areas within S_t time steps. Figure 10.1 shows this idea for several different target velocities. We also are able to observe the most important fact from this figure.

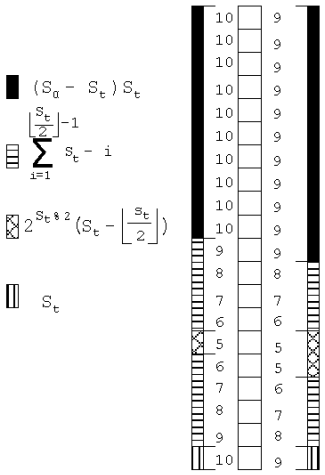


Figure 10.1: Examples of Bouncy derivation. $L = 200, r_t = 5$. Left: $v_t = 1.0$. Right: $v_t = 2.2$.

First there are always exactly $(S_\alpha - S_t)$ initial places where for each time step in S_t we

cover a new area. Therefore, we are covering precisely $(S_\alpha - S_t)S_t$ new places for these initial spots. We also note that each column finishes with a S_t. The only challenge remaining is the interior section between these two other areas.

We see that the interior area begins decreasing until it reaches a certain point, and depending on whether or not it is odd either has one or two of these lower extremes. Otherwise the decreasing is symmetric to an increasing after the lower point. Therefore if we can describe one side of the middle section, we have both sides. We can observe that

$$\sum_{i=1}^{\lfloor \frac{S_t}{2} \rfloor - 1} (S_t - i)$$

This equation covers the decreasing portion until the lowest point without adding it at all. In order to handle the lowest point we have the following equation

$$2^{S_t \% 2} \left(S_t - \left\lfloor \frac{S_t}{2} \right\rfloor \right)$$

Now putting all of the observations together we arrive at one equation that is able to count all of the newly covered squares in order to determine the average number of newly covered areas.

$$(S_\alpha - S_t)S_t + 2 \sum_{i=1}^{\lfloor \frac{S_t}{2} \rfloor - 1} (S_t - i) + \left(2^{S_t \% 2} \left(S_t - \left\lfloor \frac{S_t}{2} \right\rfloor \right) \right) + S_t \tag{10.13}$$

Although this has resulted in one equation, we notice that it isn't as simple as it could be. The modulus operator can be removed since we are simply dealing with two cases. It would also be nice to remove the summation located inside as well. For the finishing portions of the derivation we will be focusing on these two issues. We will start by trying to remove the more difficult of the two parts, the summation.

$$\begin{aligned} 2 \sum_{i=1}^{\lfloor \frac{S_t}{2} \rfloor - 1} (S_t - i) &= 2 \left(\sum_{i=1}^{\lfloor \frac{S_t}{2} \rfloor - 1} S_t - \sum_{i=1}^{\lfloor \frac{S_t}{2} \rfloor - 1} i \right) \\ &= 2 \left[S_t \left(\left\lfloor \frac{S_t}{2} \right\rfloor - 1 \right) - \frac{1}{2} \left(\left\lfloor \frac{S_t}{2} \right\rfloor - 1 \right) \left(\left\lfloor \frac{S_t}{2} \right\rfloor \right) \right] \end{aligned} \tag{10.14}$$

Now that we have arrived at a simpler equation we still need to remove the floors from the equation. There are two cases to consider. The first is the case where S_t is even.

$$\begin{aligned}
\left\lfloor \frac{S_t}{2} \right\rfloor &= \frac{S_t}{2} \\
2\left[S_t\left(\left\lfloor \frac{S_t}{2} \right\rfloor - 1\right) - \frac{1}{2}\left(\left\lfloor \frac{S_t}{2} \right\rfloor - 1\right)\left(\left\lfloor \frac{S_t}{2} \right\rfloor\right)\right] &= \\
2\left[\left(\frac{{S_t}^2}{2} - S_t\right) - \frac{1}{2}\left(\frac{{S_t}^2}{4} - \frac{S_t}{2}\right)\right] &= \\
\left({S_t}^2 - 2S_t\right) - \left(\frac{{S_t}^2}{4} - \frac{S_t}{2}\right) &= \\
\left(\frac{3}{4}{S_t}^2 - \frac{3}{2}S_t\right) &
\end{aligned} \tag{10.15}$$

The second case is when S_t is odd. Recall that we are trying to show that regardless of being odd or even we can arrive at a simple equation. Therefore, if we are able to arrive at the same equations for both even and odd cases, then we are successful.

$$\begin{aligned}
\left\lfloor \frac{S_t}{2} \right\rfloor &= \frac{S_t - 1}{2} \\
2\left[S_t\left(\left\lfloor \frac{S_t}{2} \right\rfloor - 1\right) - \frac{1}{2}\left(\left\lfloor \frac{S_t}{2} \right\rfloor - 1\right)\left(\left\lfloor \frac{S_t}{2} \right\rfloor\right)\right] &= \\
2\left[S_t\left(\frac{S_t - 1}{2} - 1\right) - \frac{1}{2}\left(\frac{S_t - 1}{2} - 1\right)\left(\frac{S_t - 1}{2}\right)\right] &= \\
2\left[\left(\frac{{S_t}^2 - S_t}{2} - S_t\right) - \frac{1}{2}\left(\frac{S_t - 1}{2} - 1\right)\left(\frac{S_t - 1}{2}\right)\right] &= \\
2\left[\left(\frac{{S_t}^2 - 3S_t}{2}\right) - \frac{1}{2}\left(\frac{{S_t}^2 - 2S_t + 1}{4} - \frac{2(S_t - 1)}{4}\right)\right] &= \\
2\left[\left(\frac{S_2^2 - 3S_t}{2}\right) - \frac{1}{2}\left(\frac{{S_t}^2 - 4S_t + 3}{4}\right)\right] &= \\
\left({S_t}^2 - 3S_t\right) - \frac{1}{4}\left({S_t}^2 - 4S_t + 3\right) &= \\
\left(\frac{3}{4}{S_t}^2 - 2S_t - \frac{3}{4}\right) &
\end{aligned} \tag{10.16}$$

As you can see we didn't arrive at the exact equation but we are very close. We will now

examine the second part of the interior of the equation. Specifically the part that holds the modulus operator. Once again we have two cases to consider. The first case to consider is when S_t is even.

$$\left(2^{S_t\%2}S_t - \left\lfloor \frac{S_t}{2} \right\rfloor\right) = \left(S_t - \frac{S_t}{2}\right)$$

And the second case is when S_t is odd.

$$\left(2^{S_t\%2}S_t - \left\lfloor \frac{S_t}{2} \right\rfloor\right) = 2\left(S_t - \frac{S_t - 1}{2}\right)$$

Now putting this all together we will arrive at our final equation. The first case considered will be the even case as always.

$$\left(S_\alpha - S_t\right) S_t + \left(\frac{3}{4}{S_t}^2 - \frac{3}{2}S_t\right) + \left(S_t - \frac{S_t}{2}\right) + S_t$$

$$\left(S_\alpha - S_t\right) S_t + \frac{3}{4}{S_t}^2$$

$$S_t S_\alpha - \frac{1}{4}{S_t}^2$$

The next case is the case where S_t is odd.

$$\left(S_\alpha - S_t\right) S_t + \left(\frac{3}{4}{S_t}^2 - 2S_t - \frac{3}{4}\right) + 2\left(S_t - \frac{S_t - 1}{2}\right) + S_t$$

$$\left(S_\alpha - S_t\right) S_t + \frac{3}{4}{S_t}^2 - S_t - \frac{3}{4} + 2S_t - S_t + 1$$

$$\left(S_\alpha - S_t\right) S_t + \frac{3}{4}{S_t}^2 + \frac{1}{4}$$

$$S_t S_\alpha - {S_t}^2 + \frac{3}{4}{S_t}^2 + \frac{1}{4}$$

$$S_t S_\alpha - \frac{1}{4}{S_t}^2 - \frac{1}{4}$$

As you can see both equations simplify out until we have very similar equations for both. In fact the only difference is a $\frac{1}{4}$ extra piece lying around on the odd side. To fix this we could always take the floor of the last piece or just ignore it. Either appears to

work fine. Recall that this calculation is the total number of freshly covered areas based on each possible starting positions. We are after the average regardless of position, therefore we divide our answer by S_α, and since finally we are equally likely to be in any of these positions we divide by another S_α.

So, our final equation is:

$$\frac{S_t S_\alpha - \frac{{S_t}^2}{4}}{{S_\alpha}^2} \tag{10.17}$$

To conclude, as with the prior controller, $S_\alpha = L/d$ and $S_t = d/v_t$. Assuming a square target controller, then (with one asset) the number of targets detected is:

$$T, \quad \text{if } S_t > 2S_\alpha \tag{10.18}$$

$$\frac{T\left[S_t S_\alpha - {S_t}^2/4\right]}{{S_\alpha}^2}, \quad \text{if } S_t \leq 2S_\alpha \tag{10.19}$$

Chapter 11

Summary

11.1 Discussion

In order to study multi-asset surveillance in a methodical manner, we have created a sophisticated simulation tool called "SURVE". SURVE has several modules, including forest generation, target controllers, asset controllers, a genetic algorithm, and modules for data collection. It can be run in either of two modes. In learning mode the genetic algorithm can be used to optimize parameters that are part of some asset controller. In performance mode SURVE collects data on the performance of the optimized controller. Our current implementation is in Java, to allow for cross-platform compatibility.

This report summarizes progress to date on optimizing potentials for the task of surveillance of multiple assets in an environment containing targets of interest and areas of foliage and non-foliage. We have concentrated on two potentials of interest: (1) the "classic" AP potential and, (2) the Lennard-Jones potential. The Lennard-Jones potential, when suitably optimized, provides superior performance in comparison with the AP potential. Furthermore, the LJ potential is robust with respect to changes in the amount of foliage, the amount of assets, and the quality of the target detector.

Finally, this report also summarizes initial mathematical analysis on a suite of simple behavior-based asset controllers.

Chapter 12

Supplemental Information

12.1 Papers Published

Spears, W., D. Zarzhitsky, S. Hettiarachchi, W. Kerr., ‘‘Strategies for Multi-Asset Surveillance.’’ 2005 IEEE International Conference on Networking, Sensing and Control, pages 929-934.

12.2 Participating Personnel

William M. Spears (Associate Professor)

Diana F. Spears (Associate Professor)

Dimitri Zarzhitsky (Ph.D. student)

Suranga Hettiarachchi (Ph.D. student)

Wesley Kerr (Masters student)

Part II

Appendix

Chapter 13

IEEE ICNSC'05 Paper

Strategies for Multi-Asset Surveillance

William M. Spears
Dimitri Zarzhitsky
Suranga Hettiarachchi
Wesley Kerr

Computer Science Department,
University of Wyoming, Laramie, WY, 82071, USA
wspears@cs.uwyo.edu
http://www.cs.uwyo.edu/∼wspears

Abstract

This paper describes our "sandbox" for the study of multi-asset surveillance, and explores the performance of rule-based control strategies on this task. In order to maximize the probability of detection of targets of interest, it is assumed that the team of unmanned air vehicles (UAVs) must provide maximum sensory coverage of the terrain. We demonstrate, however, both through simulation and mathematical analysis, that this is not always the case.

13.1 Introduction

The focus of our research is to design and build rapidly deployable, scalable, adaptive, cost-effective, and robust networks of autonomous distributed assets. The general purpose for deploying tens to hundreds of such assets can be summarized as "volumetric control." Volumetric control means monitoring, detecting, reporting, and responding to environmental conditions within a specified physical region.

In this paper we concentrate on the task of multi-asset surveillance. We assume there are α UAV assets flying at a constant altitude over terrain that contains areas of forest and non-forest. The assets have sensors to detect targets of interest, and they can determine whether they are over areas of non-forest or forest. The target sensor cannot penetrate the foliage – thus, regions of forest are of lower interest than those without forest. Each asset has a target sensor with a field of view of $\pi {r_t}^2$, and may also have a foliage sensor with a field of view of $\pi {r_f}^2$, where $r_f > r_t$. The total area of the region is much greater than $\alpha\pi {r_t}^2$. The target sensor also has a probability of detection P_d (which is assumed to be 1.0 in this paper). The goal is to locate T targets as reliably as possible, within some time period t. Targets can be stationary or mobile. When they move they can take advantage of the forest to provide cover for themselves. Our principal focus will be on whether providing maximum sensory coverage of the terrain implies maximum target detection.

13.2 Surveillance Sandbox

In order to study multi-asset surveillance in a methodical manner, we have created a sophisticated simulation tool called "SURVE". SURVE has several modules, including forest generation, target controllers, asset controllers, a genetic algorithm, and modules for data collection. It can be run in either of two modes. In learning mode the genetic algorithm can be used to optimize parameters that are part of some asset controller. In performance mode SURVE collects data on the performance of the optimized controller. Our current implementation is in Java, to allow for cross-platform compatibility.[1]

13.2.1 Forest Generation

Our "world" is a square with sides of length L, and L^2 discrete grid points. Grid point (i,j) can contain a tree or remain barren. We designed the algorithm to mimic a real forest, such that there are large clusters of trees, yet there can be holes in the middle. These holes allow assets to view targets.

The algorithm begins by placing an initial number of trees in the environment, and then uses the natural process of "seeding" in order to generate the next phase of the forest. The process continues until the desired amount of forest is reached. The controlling parameters of the algorithm are:

- n_f - The initial number of trees in the forest.
- R_s - The distance that seeds can travel.
- S_s - The amount of seeds spread during seeding.
- d_s - The decay rate of the seeds.
- F_c - The desired forest coverage.

The algorithm begins by initializing n_f trees. The placement of these trees is uniformly distributed about the world. Rather than just initializing one square for a clump of trees,

[1]The learning aspect of SURVE will not be discussed in this paper.

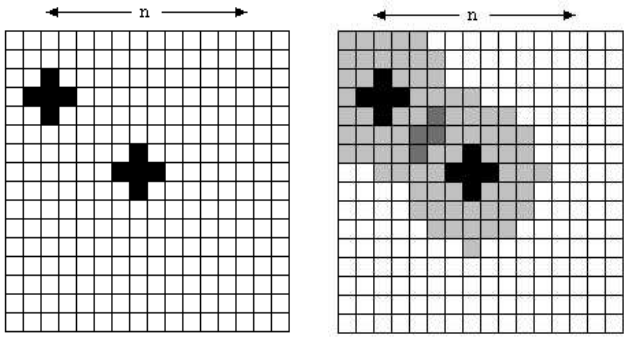


Figure 13.1: Forest seeding algorithm. *(a)* The initial forest configuration. Notice that the trees do not represent exactly one square. *(b)* The seeding portion of the algorithm. Notice that the overlap receives more seeds and is more likely to create a new tree.

we initialize a cross section, to aid in the development of the forest, see Figure 13.1(a). Once the initial trees have been placed, the next generation begins. For a given generation the current trees in the population are allowed to seed the areas surrounding them with a distance of R_s. The amount of seed laid at each position within this circle is defined as S_s. The seeding process can be seen in Figure 13.1(b). Once everything has been seeded, we begin selecting positions (i,j) that are seeded and determine if they become trees, by selecting a uniformly distributed random number. If the value selected is less than that of the seeds in the area, then the seeds have taken hold, and grow to become trees. Using this procedure not all seeds will become trees; therefore, we apply a decay to those seeds that have not become trees. Notice that as more trees surround an area, the greater the probability of growing a tree, since it has more seeds. We continue the algorithm until the desired forest coverage F_c has been achieved.

Setting the parameters is relatively easy, knowing that at full seed strength, we will have perfect circles of trees of size R_s. As we decrease S_s we begin getting more sparsely populated forests because our seeds are not as hearty and don't survive. The decay rate provides a mechanism to remove old seeds from the population of seeds, again protecting us from realizing perfect circled forests. Figure 13.2 provides a nice illustration of our forest generator. In this paper our world is 200×200 in size.

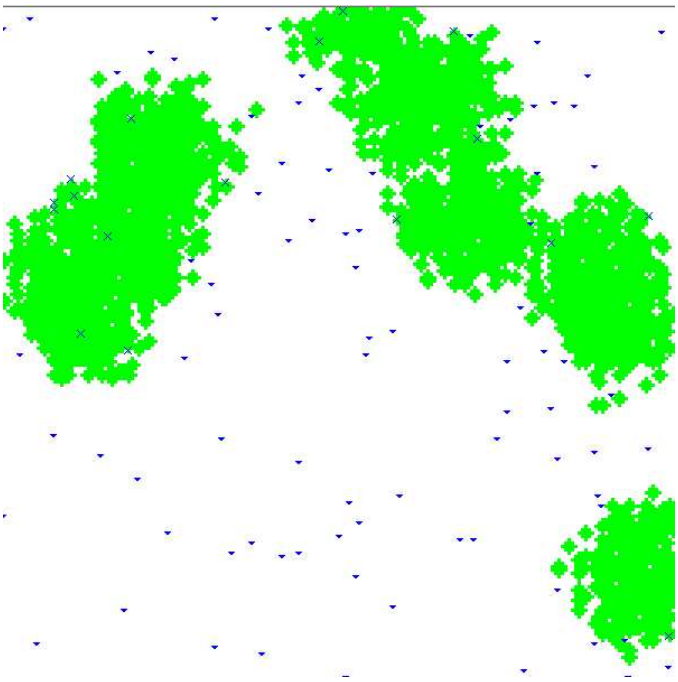


Figure 13.2: An example forest with 100 targets of interest.

13.2.2 Target Controllers

Targets can either be stationary or mobile. When they move they try to cross the environment by moving from the left to the right. We are currently creating controllers modeled after real-life scenarios, as found in the Department of the Army's Field Manual "Tank and Mechanized Infantry Company Team" (12). In this paper we will focus on one target controller, called "Gollum".

Gollum is a "sneaky" target controller that attempts to cross the environment by greedily (and locally) choosing a path with maximum foliage, so that the time a target spends exposed to the assets' sensors is minimized. To achieve the effect, each target has a foliage sensor with two attributes, a foliage radius and a foliage angle, which define how far and how wide the foliage sensor scans the environment ahead of the target to find foliage. In the absence of foliage, each target will move horizontally toward the right edge. However, when foliage is detected, the target will alter its course to reach the foliage cover. Once in the foliage, the target will again continue moving toward the right edge, and attempt to stay within the foliage as long as possible. In this paper, the target foliage sensor radius is 10,

and the foliage sensor angle is 60°, so that the sensor scans +30° and −30° away from the horizontal. These values yield a good trade-off between seeking foliage cover and avoiding large detours in target paths.

Figure 13.2 shows areas of forest, and 100 targets. The triangle represents a target that has not yet been seen but is visible, and the "×" represents a hidden target that has not been seen. The simulation also uses a "+" to represent a target that has been seen and is visible, and a "|" to represent a target that is currently hidden but has been previously seen (the latter two are not shown in this figure).

13.2.3 Rule-Based Asset Controllers

We have implemented three rule-based asset controllers. The first, called "Straight Line" (SL), is our simple base controller that moves in a straight line at velocity v. When the environment wall is hit, the asset rebounds such that the angle of reflection equals the angle of incidence. This extremely simple asset controller does not have a foliage sensor.

To illustrate the behavior of the SL controller, we monitored the "asset map" achieved by this asset strategy. This map shows locations that were seen by at least one of the assets during the given surveillance period. Areas with frequent sensor coverage are denoted by bright white areas, and areas that were observed less frequently are shown in darker hues, with black areas receiving no target sensor coverage at all. Since the target sensor can not penetrate foliage, the forested areas are also black, regardless of whether assets flew over those areas. The sample environment used for this set of experiments is shown in Figure 13.2.

As can be seen in Figure 13.3, SL-controlled assets can achieve uniform world coverage, given a moderate number of assets (ten assets appears to be enough to achieve this effect for many sets of initial conditions). However, one weakness is that assets spend time over foliage, which is not productive.

As a consequence we also implemented a "Straight Line Avoid Forest" (SLAF) controller. This controller extends SL by presuming the presence of a foliage sensor, which allows the asset to avoid spending valuable surveillance time over forested areas, where target sensors are ineffective. The only information that is assumed to be available from this simple

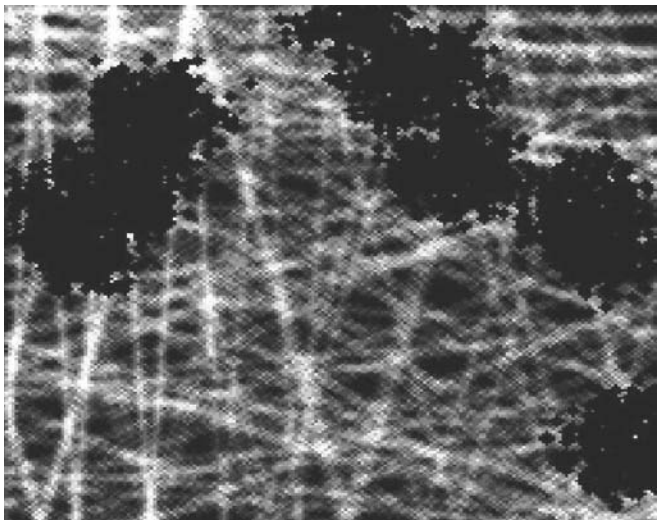


Figure 13.3: Sensor coverage with the SL strategy after 1,000 steps.

foliage sensor is the percentage of the sensor view field occupied by the foliage. Once that percentage exceeds a certain threshold (currently set at 50%), the asset will reverse its direction and will move away from the edge of the forest. Should the asset be initially deployed directly over the foliage, it will continue moving in its initial direction until it leaves the foliage, or an interaction with a world boundary causes it to turn around; in either case, given sufficient time, the asset will end up in a foliage-free location.

As can be seen in Figure 13.4, SLAF-operated assets frequently move in a short, localized pattern, and fail to achieve uniform sensor coverage regardless of the amount of time the surveillance activity is carried out.

To address the poor coverage of SLAF, we finally implemented a "Super Straight Line Avoid Forest" (SSLAF) controller that utilizes a more sophisticated foliage sensor. Upon detecting an increased amount of foliage within its sensor (current threshold is set at 10%), the asset will enter an "avoid forest" state, computing the mass distribution of the sensed foliage and moving in the direction exactly opposite the center of foliage mass. Once the detected foliage drops below the threshold, the asset will switch into its normal surveillance state, continuing to move in its current direction until an encounter with another patch of foliage or a bounce from an environment boundary occurs. This strategy was inspired by Reynolds (8) and Balch (2).

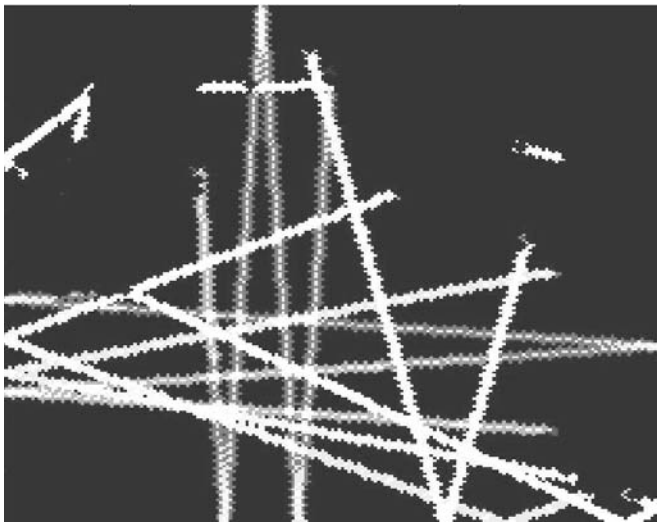


Figure 13.4: Sensor coverage with the SLAF strategy after 1,000 steps.

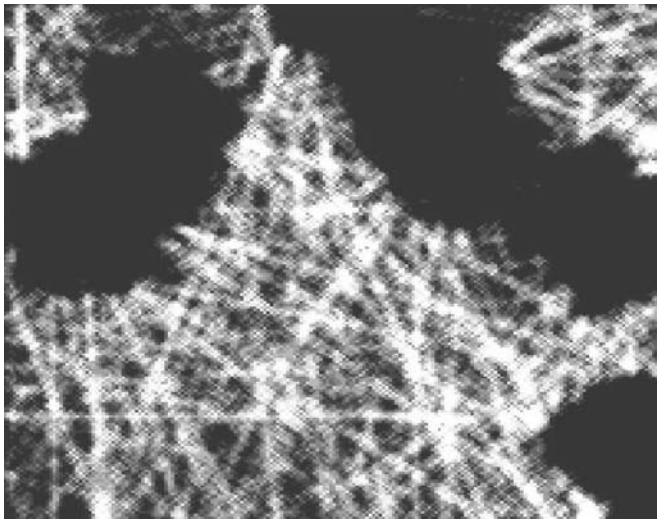


Figure 13.5: Sensor coverage with the SSLAF strategy after 1,000 steps.

As can be seen in Figure 13.5, SSLAF-controlled agents tend to achieve a far more more comprehensive sensor coverage than SLAF, and they achieve this faster than do SL-based assets (as indicated by the brighter coverage of SSLAF versus SL).

13.3 Results with Stationary Targets

We first tested the SL, SLAF, and SSLAF asset controllers with environments containing 100 stationary targets. The target radius $r_t = 2$, while the foliage radius $r_f = 5$. Regardless of controller, all assets moved at speed $v = 3$. The percentage of foliage varied from 0% to 70% in increments of 10%. The performance metric is the percentage of targets spotted within $t = 200$ times steps, given that these targets were in fact visible. For each percentage of foliage 100 forests were randomly generated, and for each of the forests the results were averaged over 50 runs with different target and initial asset placements.

Table 13.1: Percentage of Stationary Targets Found (10 assets)

Foliage (%)	SL	SLAF	SSLAF
0	41.18	41.18	41.18
10	39.05	22.47	39.87
20	41.25	20.09	41.11
30	39.25	17.43	43.12
40	38.98	15.83	44.53
50	40.67	17.73	50.19
60	41.35	18.19	54.57
70	39.98	20.13	57.58

Tables 13.1 and 13.2 give the performance of the three asset controllers, with $\alpha = 10$ and $\alpha = 30$ assets. With 0% foliage, all controllers perform equivalently, since the foliage sensors have no function to perform. Also, the performance of SL is independent of the percentage of foliage, since it has no foliage sensor. In general, SSLAF outperforms SL, especially as the percentage of foliage increases. This is a consequence of the fact that SL wastes time over areas of foliage. Finally, as expected, SLAF performs poorly, due to its lack of ability to provide uniform coverage of the environment.

Table 13.2: Percentage of Stationary Targets Found (30 assets)

Foliage (%)	SL	SLAF	SSLAF
0	79.52	79.52	79.52
10	79.71	54.30	75.73
20	79.33	48.07	77.67
30	78.14	44.06	77.63
40	78.54	43.01	79.55
50	78.86	43.78	81.25
60	79.07	44.22	84.05
70	77.27	47.46	84.49

13.4 Results with Moving Targets

For the next set of experiments we allowed the targets to move according to their "Gollum" controller. Target speed was 0.4, with all other experimental variables fixed as before.

Table 13.3: Percentage of Gollum Targets Found (10 assets)

Foliage (%)	SL	SLAF	SSLAF
0	32.28	32.28	32.28
10	29.78	29.63	31.35
20	25.46	26.50	29.04
30	23.09	25.60	28.47
40	18.59	22.72	24.02
50	17.45	23.10	24.85
60	14.54	21.93	22.39
70	13.61	23.11	23.10

Tables 13.3 and 13.4 give the performance of the three asset controllers, with $\alpha = 10$ and $\alpha = 30$ assets. Again, with 0% foliage, all controllers perform equivalently, since the foliage sensors have no function to perform. However, in general, the task is clearly more difficult. This is a consequence of the Gollum controller, where targets actively seek out foliage for cover. As expected, SSLAF still outperforms SL. However, the results with SLAF look anomalous. With moving targets SLAF performs roughly equivalently to SSLAF (and

Table 13.4: Percentage of Gollum Targets Found (30 assets)

Foliage (%)	SL	SLAF	SSLAF
0	65.02	65.02	65.02
10	64.66	64.10	65.57
20	55.55	56.25	57.27
30	51.13	54.79	54.52
40	45.01	51.05	50.55
50	39.50	48.46	47.36
60	36.61	48.47	46.67
70	30.78	45.75	40.90

in fact outperforms SSLAF when there are a large number of assets and the percentage of foliage is high)!

Recall that our initial motivation for creating the behavior of our asset controllers was predicated on the belief that maximum sensor coverage of the domain implies maximum target detection. For stationary targets, this belief was validated. However, as soon as the targets start to move (with a speed that is quite slow relative to the asset speed), an asset controller with extremely poor domain coverage provides very good target detection. The next section provides a mathematical analysis of why this occurs.

13.5 Analysis

Informally, one can start to understand the error in our belief if one notices that the belief was grounded in spatial reasoning. Clearly, for stationary targets, uniform coverage of the space is necessary (if one assumes the targets also are uniformly distributed throughout the space). The asset map shown in Figure 13.4 clearly indicates that vast portions of the domain are unexplored, leading to poor target detection.

However, once targets move, spatial reasoning must be combined with temporal reasoning. To illustrate this, Figure 13.6 shows a "target map" illustrating the movement of the Gollum targets over time, as the targets move from left to right through the environment. The passage of targets through foliage are not shown, since they are not visible to the assets during that time. One can notice that a clump of foliage provides a "focusing" effect that

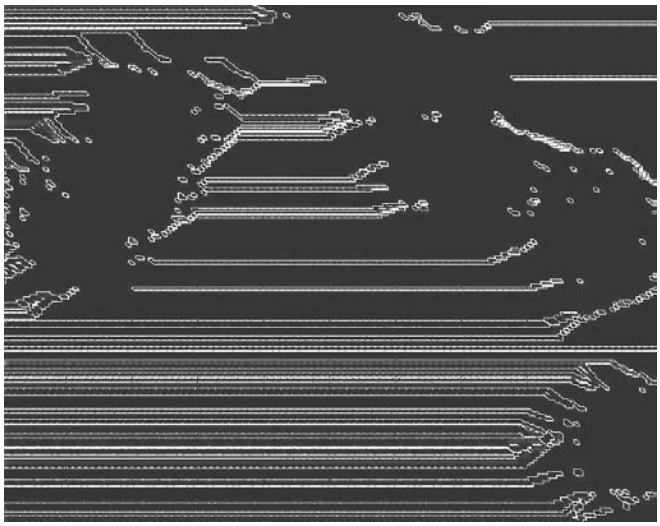


Figure 13.6: Gollum target coverage over 1,000 steps.

changes the density of targets (along a column) from uniform to highly non-uniform.

For the sake of target detection, it can be seen that if one mentally overlays Figures 13.4 and 13.6, all that is really required for detection is an intersection between the target paths and the asset paths, *at the same moment in time*. Although the target and asset maps provide spatial information, their intersection in time is not represented. For this we will require some probabilistic analysis.

First, for the sake of tractability, we assume that the $L \times L$ environment contains no forest, and that T targets are crossing the environment horizontally from left to right. We consider four situations: (A) an asset is bouncing back and forth in the domain along the horizontal, (B) an asset is bouncing along the vertical, (C) an asset is bouncing along the major diagonal, and (D) an asset is bouncing in such a way as to uniformly cover the space. We will then compute the *expected number of targets* that the asset will detect. Note that the situations are specifically designed to differentiate between a highly uniform coverage of the space versus a highly non-uniform coverage.

13.5.1 Horizontal Movement

This situation is the easiest to analyze. If one asset has target radius r_t, then it will sweep a row with height $2r_t$. If T targets are uniformly distributed along the left-most column as

they start their movement, then the asset will detect $T2r_t/L$ targets. If there are α assets with non-overlapping horizontal paths, then the expected number of targets detected is:

$$\frac{\alpha T 2 r_t}{L} \tag{13.1}$$

Note that the velocity of the asset is not relevant for this particular situation. The velocity of the target must be greater than zero.

13.5.2 Vertical Movement

Again, assume the asset has target radius r_t. It will sweep a column of width $d = 2r_t$. Assume for the sake of simplicity that the velocity of the asset $v = 2r_t$, so that overlap does not occur. Then the number of steps required for the asset to traverse the column $S_\alpha = L/d - 1$. Thus, any grid point in the column is hit with a probability approximated by $1/S_\alpha$, and is not hit with probability approximated by $(S_\alpha - 1)/S_\alpha$. Finally, we need to calculate how long a *target* will be within that column. If the speed of the target is v_t, then the target will require $S_t = d/v_t$ steps to cross the column. Hence the probability of that target not being detected is approximately $((S_\alpha - 1)/S_\alpha)^{S_t}$. Finally, the expected number of targets detected is approximately:

$$T\left[1 - \left(\frac{S_\alpha - 1}{S_\alpha}\right)^{S_t}\right] \tag{13.2}$$

If there are α independent assets then the number of targets detected is:

$$T\left[1 - \left(\frac{S_\alpha - 1}{S_\alpha}\right)^{\alpha S_t}\right] \tag{13.3}$$

13.5.3 Diagonal Movement

This situation is a small transformation of vertical movement. Since moving along the diagonal is a factor of $\sqrt{2}$ longer than moving along the column, both S_α and S_t must be renormalized. In this situation $S'_\alpha = \sqrt{2}L/d - 1$ and $S'_t = \sqrt{2}d/v_t$. If there are α independent assets then the number of targets detected is:

$$T\left[1-\left(\frac{S'_\alpha-1}{S'_\alpha}\right)^{\alpha S'_t}\right] \tag{13.4}$$

13.5.4 Uniform coverage

Finally, if an asset is uniformly covering the domain then $S''_\alpha = L^2/\pi {r_t}^2$ and $S''_t = L/v_t$. If there are α independent assets then the number of targets detected is:

$$T\left[1-\left(\frac{S''_\alpha-1}{S''_\alpha}\right)^{\alpha S''_t}\right] \tag{13.5}$$

13.5.5 Theory versus Simulation

For confirmation of the theory, we ran SURVE with 100 Gollum targets, no forest, asset speed $v = 10$, target radius $r_t = 5$, and target speed $v_t = 0.4$. The number of assets were one, two, and four.

Table 13.5: Expected Number of Targets Detected

α	Horizontal	Vertical	Diagonal	Uniform
1	5	74	73	62
2	10	93	92.9	86
4	20	99.6	99.5	98

Table 13.6: Actual Number of Targets Detected

α	Horizontal	Vertical	Diagonal	Uniform
1	5	73	77	63
2	10	91	93	86
4	18	98	99	98

Table 13.5 shows the theoretical results from the above equations, for the four situations. Table 13.6 shows the empirical results, averaged over 1000 runs. The empirical results agree very well with the theoretical results, and are quite informative. First, as might be

expected, horizontal movement of the asset yields the poorest performance (assuming that targets are also moving horizontally). However, column movement and diagonal movement definitely outperform uniform coverage! Hence, although uniform coverage is excellent for stationary targets, this strategy is suboptimal for moving targets. If one re-examines the asset map shown in Figure 13.4, one can clearly see the long vertical movements that are yielding the good performance of SLAF. The diagonal asset control strategy is especially interesting, since it will work well regardless of whether targets cross the domain horizontally or vertically.

13.6 Summary

This paper presents the SURVE toolkit for experiments in multi-asset surveillance. SURVE has several modules, including forest generation, target controllers, asset controllers, a genetic algorithm, and modules for data collection. Due to the efficiency of implementation, SURVE can be run with hundreds of assets and thousands of targets. We then present evidence that although the goal of uniform coverage of a domain is excellent for detecting stationary targets, it is suboptimal for moving targets. Theoretical analysis confirms the empirical results, indicating that specialized patterns of surveillance will outperform more generalized uniform coverage.

Prior work in this area includes Wu et. al., who used the SAMUEL learning system to evolve rule sets that control a set of micro-air vehicles (MAVs) to provide maximum coverage of a region (13). Wu expanded on this by combining chunking with a GA to increase performance (14). Independently, Bugajska combined SAMUEL with a GA to find an optimal sensor suite and reactive rules (3), and combined SAMUEL with ACT-R to provide a cognitive model (4). Sukhatme et. al. have used a behavior-based approach to surveillance with cooperative aerial and ground vehicles (11). Albekord et. al. summarize a multi-tiered control strategy for multiple-asset surveillance, utilizing both air and ground vehicles (1). Spears et. al. discuss a physics-based distributed algorithm for controlling swarms of UAVs for surveillance (9; 10). Pack and Mullins propose a method for searching an area according to four basic search rules, in order to provide complete coverage of the domain (7). Finally, Krishna et. al. provide a surveillance system based on multiple mobile

sensors (6). Interestingly, this latter paper uses an "X" formation of assets, where the assets are placed along both major diagonals of the environment. However, these papers do not provide theoretical justification for the behavior of the assets.

We are currently attempting to generalize our theoretical analysis to include forest distributions. The eventual goal is to formally merge the spatial asset and target maps into a temporal construct that is predictive of target detection probability, allowing us to construct optimal asset strategies for given target controllers.

Acknowledgment

Thanks to Diana F. Spears for suggesting the use of asset and target maps for visualization of the system.

BIBLIOGRAPHY

[1] K. Albekord, A. Watkins, G. Wiens, and N. Fitz-Coy, *Multiple-Agent Surveillance Mission with Non-Stationary Obstacles*, Florida Conference on Recent Advances in Robotics, 2004.

[2] T. Balch and R. Arkin, *Behavior-based Formation Control for Multi-Robot Teams*, IEEE Transactions on Robotics and Automation, 1998 (14) 1–15.

[3] M. Bugajska and A. Schultz, *Co-Evolution of Form and Function in the Design of Autonomous Agents: Micro Air Vehicle Project*, Workshop on Evolution of Sensors, Genetic and Evolutionary Computation Conference, 2000, 240–244.

[4] M. Bugajska, A. Schultz, J. Trafton, S. Gittens and F. Mintz, *Building Adaptive Computer Generated Forces: The Effect of Increasing Task Reactivity on Human and Machine Control Abilities*, Genetic and Evolutionary Computation Conference Late Breaking Papers, 2001, 24–29.

[5] D. Goldberg and M. Mataric, *Design and Evaluation of Robust Behavior-Based Controllers*, Chapter 13 of Robot Teams (eds. T. Balch and P. Parker): A.K. Peters, 2002, 369–380.

[6] K. Krishna, H. Hexmoor, P. Rao and S. Chellapa, *A Surveillance System based on Multiple Mobile Sensors*, FLAIRS, Special Track on AI Techniques in Multi-sensor Fusion, 2004.

[7] D. Pack and B. Mullins, *Toward Finding an Universal Search Algorithm for Swarm Robots*, International Conference on Intelligent Robots and Systems, 2003, 1945–1950.

[8] C. Reynolds, *Steering Behaviors for Autonomous Characters*, Proceedings of Game De-

velopers Conference, 1999, 763–782.

[9] W. Spears, D. Spears, R. Heil, W. Kerr and S. Hettiarachchi, *An Overview of Physicomimetics*, Lecture Notes in Computer Science - State of the Art Series, 2005, (3342).

[10] W. Spears, D. Spears, J. Hamann, and R. Heil, *Distributed, Physics-Based Control of Swarms of Vehicles*, Autonomous Robots, Volume 17(2-3), 2004, 137–162.

[11] G. Sukhatme, J. Montgomery, and R. Vaughan, *Experiments with Cooperative Aerial-Ground Robots*, Chapter 12 of Robot Teams (eds. T. Balch and P. Parker): A.K. Peters, 2002, 345–367.

[12] *Tank and Mechanized Infantry Company Team*, Department of the Army Field Manual 71-1, 2002.

[13] A. Wu, A. Schultz, and A. Agah, *Evolving Control for Distributed Micro Air Vehicles*, IEEE Conference on Computational Intelligence in Robotics and Automation, 1999, 174–179.

[14] A. Wu and H. Stringer, *Learning Using Chunking in Evolutionary Algorithms*, University of Central Florida, Computer Science Department, Technical Report, 2002.

Part III

Appendix

Chapter 14

User Manual for SURVE

SURVE Toolkit: the User's Guide

William M. Spears
Diana F. Spears
Wesley Kerr
Suranga Hettiarachchi
Dimitri Zarzhitsky

Computer Science Department,
University of Wyoming, Laramie, WY, 82071, USA
wspears@cs.uwyo.edu
http://www.cs.uwyo.edu/∼wspears

14.1 Introduction

This manual covers the SURVE simulator for automated outdoor surveillance, focusing on beginner and intermediate concepts needed to install the SURVE toolbox, conduct experiments, and to collect data. The SURVE toolkit is intended to facilitate the study of a surveillance problem, where several mobile aerial *assets* attempt to detect the presence of ground-based *targets*, which may or may not be moving. To simplify the discussion, this manual assumes that the assets are uninhabited aerial vehicles (UAV), charged with detection of invading vehicles such as tanks and supply convoys, that may attempt to conceal their presence by hiding under the tree canopy or by avoiding the UAV sensors.

Thus the toolkit is a test bed for different types of surveillance problems, and is distributed with several realistic control strategies for both the assets and targets. Users can run interactive experiments, with real-time feedback via an advanced GUI, and can execute a pre-programmed set of experiments in a non-interactive batch mode. Simple simulation statistics are computed within the simulator, and raw simulation data can be stored for later post-processing and in-depth analysis.

This document first outlines how to install the SURVE toolkit and to run basic experiments in the interactive mode, and then explains how to create experiment sets for batch processing. It also provides a detailed overview of the GUI controls and the configuration files. Please keep in mind that SURVE software is still in development, and thus minor inconsistencies or unresolved issues may persist in the working version of the program.

14.2 Quick Start

The SURVE toolkit is written using the multi-platform Java2 programming language, and is distributed in the form of a ZIP and TAR archives. It should run on any platform for which a Java Runtime Environment (JRE) is available, and has been extensively tested with Linux and Windows operating environments. To install the software, simply uncompress the distribution file into an empty directory (consult your operating system manual or system administrator for assistance with this step). The distribution archive includes all of the compiled files along with the source code, and no further configuration is necessary.

To execute the SURVE simulator, change into the `surveillance2` directory that was created when you decompressed the distribution archive, and run the Java interpreter on the `graphics.simulator.Observer` class. If you're not sure how to execute Java programs on your computer, please consult the Java help page at http://java.sun.com/help/. If you use a UNIX-like operating environment, then simply execute the `runVis.sh` script from a command prompt. When you see a window similar to the one shown in Fig. 14.1 then you have successfully installed the SURVE toolbox, and can start running experiments, as outlined in the following sections.

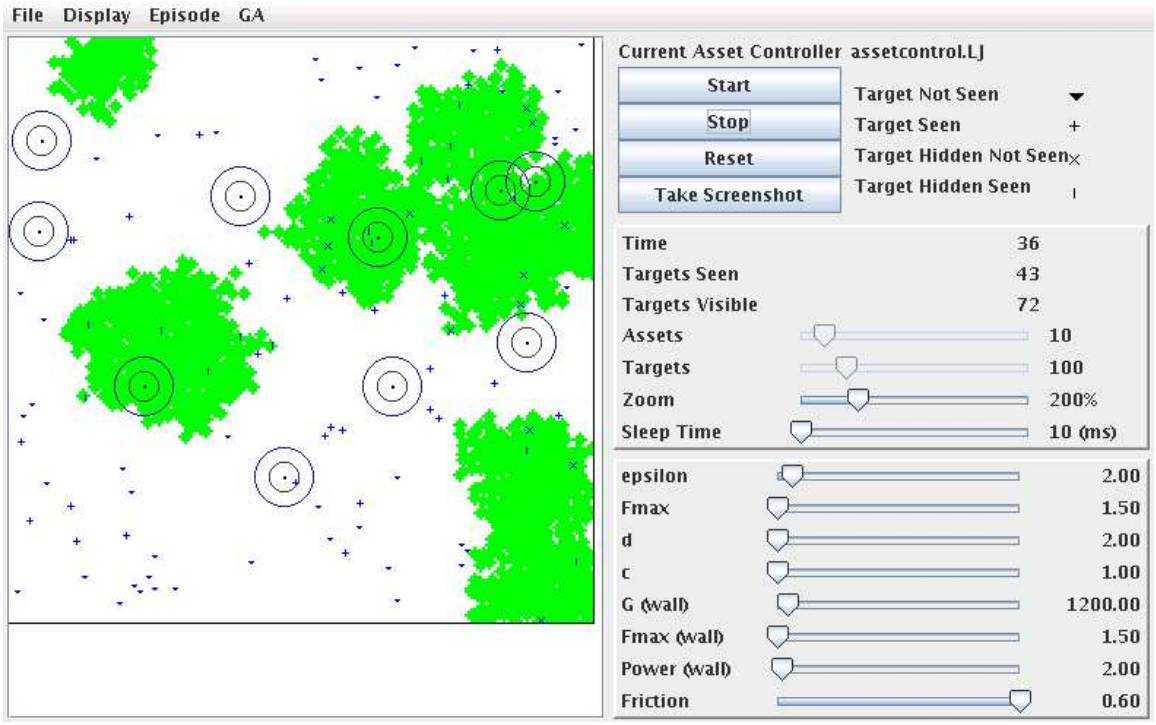


Figure 14.1: SURVE simulator window.

14.3 Surveillance Experiments

Before explaining how to use the SURVE toolkit to conduct surveillance experiments, it will be helpful to explain the methodology behind the design of the toolkit, so that it will be easy to understand the structure of the experiments. For the purposes of the toolkit, we

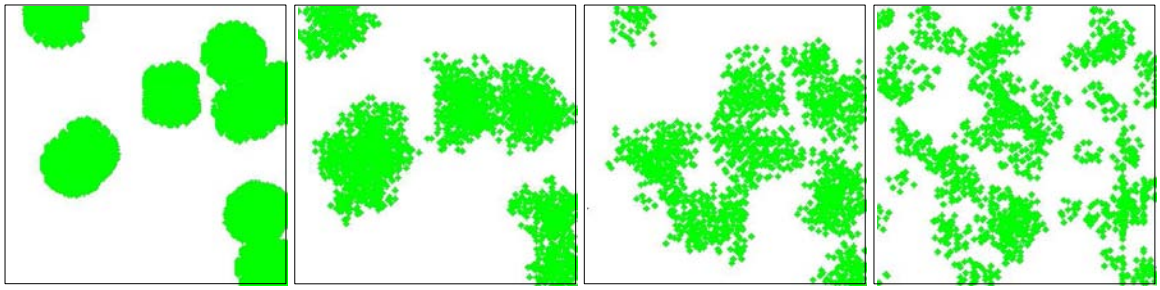
assume that surveillance is taking place in a rectangular region. The landscape is assumed to be flat, but the presence of dense forest, impenetrable by the sensors, complicates the surveillance problem. Two basic surveillance scenarios are of interest: a case where the targets are stationary (perhaps awaiting deployment), and a case where targets move in some organized, directed fashion (as may be the case of a motorized assault or a resupply mission). In both scenarios, the monitoring UAVs move in a trajectory indicated by the pre-programmed surveillance strategy for a given length of time, consistent with the fact that UAVs have a time-of-flight limit imposed by limited energy resources.

During the surveillance activity, the UAVs continually scan the ground underneath them, and whenever the sensor beam detects a target, the target is marked as "seen" by the simulator as part of its data-collection activity. There are several user-specified parameters that affect how the target sensor operates, but for clarity, the discussion of these and other configuration parameters is presented in the following sections. Some of the asset strategies distributed with the SURVE toolkit also assume that the UAVs are equipped with forest sensors, in hopes that the prudent use of extra sensor information will increase the number of detected targets in a given time period.

14.3.1 Building Simulated Environment

The first step in the surveillance scenario is the construction of the environment. Two main parameters characterize the area over which surveillance is conducted: the size, which is a single integer dimension defining a square-shaped area, and the extent of world foliage cover, specified as a floating point fraction less than one. World size is specified in the `conf/general.txt` configuration file, in the `Environment Parameters` section using the `size=`*integer* entry, where the positive integer value is the length of the world boundary in screen pixels. Keep in mind that the larger the world size, the greater the computational resources (memory and CPU power) that are needed to execute the simulator. The default world size is 200×200 units.

The foliage growth in the world creates areas where UAV target sensors are ineffective; the notion of simulated foliage may also be thought of as an abstraction of other environmental obstacles that may inhibit surveillance sensors (e.g., caves or tunnels within a mountain



seedRadius=5	seedRadius=7	seedRadius=8	seedRadius=10
seedStrength=0.5	seedStrength=0.007	seedStrength=0.02	seedStrength=0.95
seedDecay=0.5	seedDecay=0.7	seedDecay=0.98	seedDecay=0.95
forestSize=10	forestSize=7	forestSize=15	forestSize=90

Figure 14.2: Foliage distributions produced by the environment generator along with the values of the corresponding algorithm parameters. These images show a 30% total area coverage of the 200×200 world.

terrain), although the environment builder supplied with the current version of the surveillance simulator only mimics natural plant growth (e.g., a forest of trees). Environment generator has several parameters that control the type of foliage distribution produced, all of which are specified in the `conf/forest.txt` configuration file (see Fig. 14.2).

Different foliage maps can be created (and recreated) by specifying a nonnegative seed to the random number generator (look for `forestSeed=`*integer* in the `conf/forest.txt` file). The default value of -1 tells the simulator to use the current system time (in seconds) as the random seed, so that a different forest is generated each time the simulator is run, or when the forest is reset via the Episode menu (see Section 14.6 on simulator menus).

To control the foliage placement, first consider the following high-level description of the forest generation algorithm: the forest begins as a collection of randomly placed "trees", where the initial number of these starter trees is given by the `forestSize=`*integer* parameter in the `conf/forest.txt` file. Then, each tree produces "seeds" which fall to the ground surrounding a given tree within the area specified by the `seedRadius=`*integer* configuration parameter. Seed viability is controlled by the `seedStrength=`*float* value, which represents the probability that a given seed will produce a new tree. Note that a location with two seeds is twice as likely to produce a new tree than a location with just one seed, given equal seed strengths. As time passes, each seed ages and becomes less likely to grow into a tree; the rate at which the seeds weaken is specified by the `seedDecay=`*float* line in the

forest configuration file. All grown trees continue to produce new seeds until the total world foliage cover is equal to the `forestCoverage=`*float* parameter. *Note:* be careful when using small values for `seedRadius` and `seedStrength` or large values for `seedDecay` and `forestSize`, as it may be either impossible or may take a prohibitively long time to achieve the specified `forestCoverage` with seeds that are extremely weak, decay very quickly, or if the initial number of starter trees exceeds the world area; the ranges shown in Fig. 14.2 work well for many environments, and should be treated as a guide for future settings of these parameters.

14.3.2 Initializing Targets

After constructing the desired simulated environment, the next step in the experiment preparation process is the set up and initialization of targets – the objects which will be detected during the surveillance activity. Each target is assumed to have a 1×1 unit dimension, and may move with constant speed in any direction or remain stationary during the simulation. A target's movement trajectory is computed by a target controller, and currently, all targets in the simulation share the same target controller algorithm during any given run. However, a different target group behavior may be obtained quickly, without restarting the simulator, by simply switching the target controller via the File menu (see Section 14.6 on simulator menus).

In order to evaluate the performance of a given surveillance strategy, it is helpful to enumerate the relevant properties and states of the simulated targets. Each target, at any given time step, may be either visible or hidden. A visible target is located within the world boundaries at a point without foliage cover. Likewise, a hidden target is either outside of the world boundaries, or is located under the dense cover of the forest, and cannot be observed with the surveying equipment. For moving targets, the "hidden" and "visible" target states refer to the *current* state of a target, and may change many times during the simulation, unlike the "seen" and "unseen" classification, that may only change once. As the name implies, the "seen" category is used to describe targets that have been detected by a UAV target sensor, while "unseen" targets are the ones that have successfully escaped the detection. Note that all of simulated targets, including those that are stationary, start

out in the unseen group, but will change their state to seen after being detected by the UAVs. Thus, one of the most important performance metrics used to assess the merit of any surveillance strategy during an experiment is the ratio of seen targets to all of the targets that have been visible (and thus potentially detectable) at least once during the run.

It will come as no surprise that the UAV surveillance performance depends on the movement strategy employed by the targets. In other words, a surveillance approach that does well in detecting stationary targets might do poorly in detecting moving targets, and vice versa. Therefore, the SURVE simulator is distributed with several realistic target controllers that can be selected at run time:

- *Avoid:* targets attempt to actively avoid the ground-level sensor footprint of the overhead UAVs while crossing the area under surveillance. When a moving target detects a UAV sensor beam in its path, it pauses for a short time, waiting for the UAV to move away; if the sensor beam persists after the delay, then the target will attempt to circumnavigate around the sensor footprint.

- *Ribbon:* targets follow a periodically sinuous trajectory, which varies slightly for each target. The amplitude and period of the mean-path oscillation are kept low to approximate realistic deviations from straight line travel due to terrain configuration.

- *Sneaky:* targets aim to minimize their exposure time to the UAV sensors by attempting to maximize the amount of travel time spent underneath the foliage. This target controller assumes that the targets may scan their local environment for foliage-covered areas, and will select the next waypoint to be inside the forest if possible.

- *Stationary:* targets simply do not move.

- *StraightLine:* targets move in a horizontal line formation from the left world boundary to the right edge of the surveillance area.

- *militaryCombo:* targets move in a combination of wedge, vee, echelon left/right, and column formations from the left world boundary to the right boundary of the surveil-

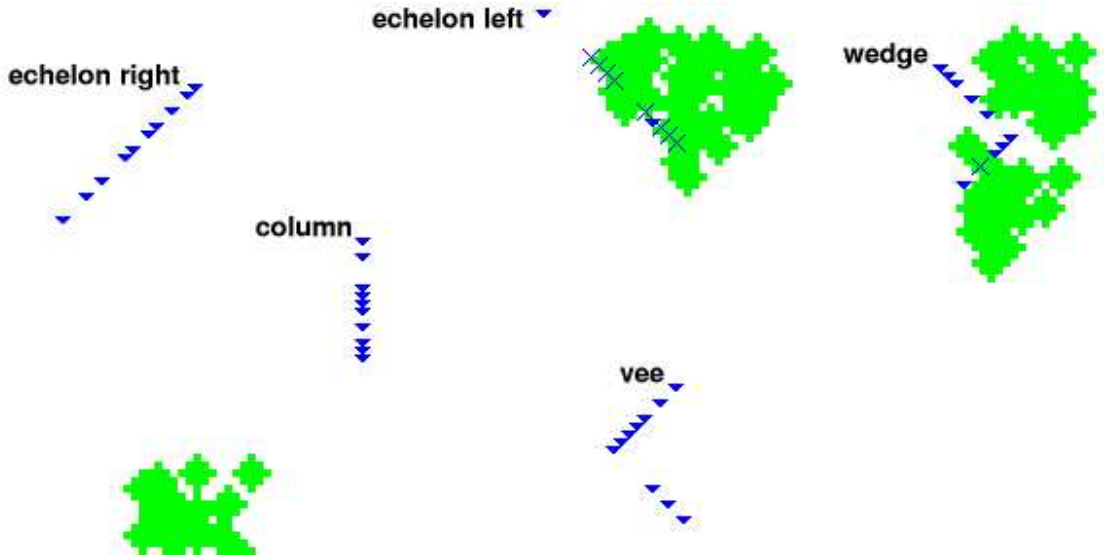


Figure 14.3: Targets arranged in the wedge, vee, echelon left/right, and column formations.

lance perimeter. These target groupings mimic the actual tank battlefield formations, and a representative sample of each formation type is shown in Fig. 14.3.

Because of the varying nature of the trajectories generated by each target controller type, some configuration parameters differ significantly for each controller choice. However, one common aspect of target initialization is that it is done at random, using either a uniform or a normal probability distribution. The random number generator seed used by the target controllers is given by the `targetSeed=`*integer* parameter in the `conf/general.txt` configuration file; the default is a negative value, which yields a different random sequence each time the simulator is invoked). The number of targets in the simulation is controlled by the `targets=`*integer* parameter, with the default value of a 100. Target speed is specified by the `targetSpeed=`*float* parameter, and the units match world dimensions, so that a `StraightLine` target with a speed of 1.0 will cross a 200×200 world in 200 simulation steps, unless collision avoidance procedures slow down the moving target. The `targetSpeed` parameter is ignored by the `Stationary` targets.

Additional control of the targets is possible through the `targetOffset=`*float* parameter, which specifies the horizontal x coordinate location of the initial target placement random distribution center point in terms of the world size. For instance, a `targetOffset=0.0`

setting places the center of the target distribution in the middle of the surveillance area, so that all targets start out within the surveillance area boundary (this should be the setting used for most experiments with the `Stationary` target controller). Setting `targetOffset` to -1.0 causes all of the targets to be initially placed before the left world edge, outside of the surveillance area, but still using the same random distribution (think of the target placement as just being shifted to the left by one world size); this is useful if the UAV units require time to organize, and a delay between the UAV deployment and the first target appearance is desired. Note that due to the fact that the standard SURVE controllers move targets from left to right, positive `targetOffset` values have little utility. The initial target controller may be selected via the `targetControl=`*string* parameter, and its value should be a fully-qualified Java class name, such as `targetcontrol.Stationary`; alternatively, the target controller may be changed via the File menu (see Section 14.6 on simulator menus).

14.3.3 Initializing Assets

Intelligent, efficient, and effective control of the surveillance UAVs is the major topic of interest driving this research. The chief aim is to devise a surveillance strategy that maximizes the target detection rate by maximizing the benefits of asset-asset collaboration, sensor data utilization, and clever exploitation of the existing domain knowledge. Thus it is of little surprise that the thrust of the SURVE development work focused on the asset behavior.

A conceptual schematic of the SURVE UAV asset model is given in Fig. 14.4, which depicts a single asset in the vicinity of some foliage, and shows the ranges of the three types of sensors assumed to be present on every UAV in the simulation. The most basic aspect of the UAV model is the asset representation, which assumes an arbitrarily small asset size (a point model) with a real-valued coordinate location. The most important UAV sensor is of course the target sensor, whose radius is specified in the `conf/asset.txt` file by the `targetRadius=`*integer* parameter.

Because of the critical role the target sensor plays in the surveillance problem additional configuration options are available for customizing target sensor characteristics. The shape of the ground-level target sensor footprint is controlled by the `targetSensorType=`*string*

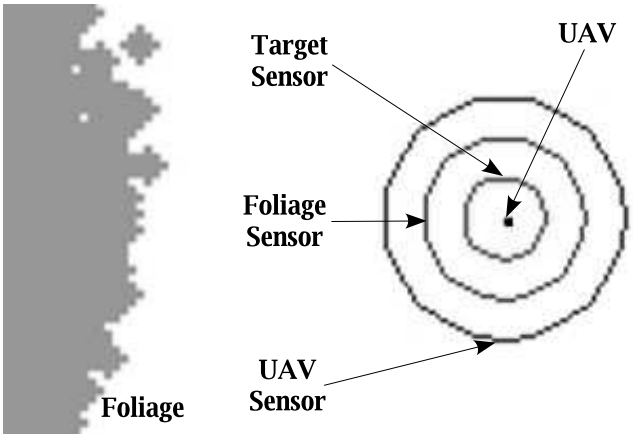


Figure 14.4: The model of a surveillance asset, showing the three on-board sensor types with their ranges.

parameter, with possible values of *circle* or *square*. In case of a *square* target sensor, the `targetRadius` parameter defines half the width of the ground-level sensor footprint, so that a setting of `targetSensorType=square` and `targetRadius=5` will produce a sensor with a 10×10 target detection area. The Boolean `targetSensorIncludeBounds=`*0|1* parameter is a fine-tuning option which controls whether the sensor will detect targets that are located right on the sensor footprint boundary. A value of 0 indicates that a strict $<$ (less than) comparison is to be used when computing distances, thus excluding targets that only touch the sensor boundary, and a value of 1 specifies that a $\leq$ (less than or equal) test be used throughout the simulation, allowing detection of such targets. Perhaps the most crucial parameter is the `detectionProb=`*float* specifier, which determines the percentage of sensor readings that correctly detect a target when it is in the view-field of the sensor. Thus a setting of 0.0 will inhibit all target detection (extreme sensor noise), and a setting of 1.0 will construct a perfect target sensor, which always properly identifies a target when the UAV passes over it.

Two other types of sensors shown in Fig. 14.4 are the asset-to-asset sensor and the foliage detector. These sensors are "optional" in the sense that only some of the built-in UAV controllers make use of them. In particular, interactions between assets are managed by controllers based on the *Artificial Physics* framework (www.cs.uwyo.edu/∼wspears/ap.html for more information). Likewise, information about foliage cover in the environment is

exploited by only a few of the asset controllers included in the default SURVE distribution (the size of the circular area covered by the foliage sensor of each UAV is set by the `foliageRadius=`*integer* parameter in the `conf/asset.txt` configuration file). The list below provides the detailed information about each of the standard asset controllers:

- *AP:* is a physics-based controller which uses a generalized form of Newton's Law of Universal Gravitation, $\vec{F} = Gm_1m_2/r^p$, to compute a virtual force between asset 1 and asset 2, with respective virtual masses of m_1 and m_2, separated by a distance r (see Fig. 14.5). In many cases, the assets are assumed to have the same virtual mass, along with some desired separation dictated by practical constraints, such that the pair-wise inter-asset force is attractive if the actual separation between UAVs exceeds the desired range, and becomes repulsive if the assets move too close to each other. By varying the parameters which define this force, a variety of dynamically stable UAV formations can be self-organized and maintained in a completely distributed fashion. The force law may be altered manually using the controls in the bottom-right window panel (see Section 14.6 on simulator controls), or alternatively, the appropriate set of values may be constructed using a Genetic Algorithm, and loaded into the simulator from the evolution history file via the GA menu (see Section 14.6 on simulator menus).

- *Bouncy:* is a controller motivated by the need to carry out statistical experiments, and simply moves the assets vertically, "bouncing" off the top and bottom world boundaries by reversing the UAV velocity upon reaching the edge of the surveillance area.

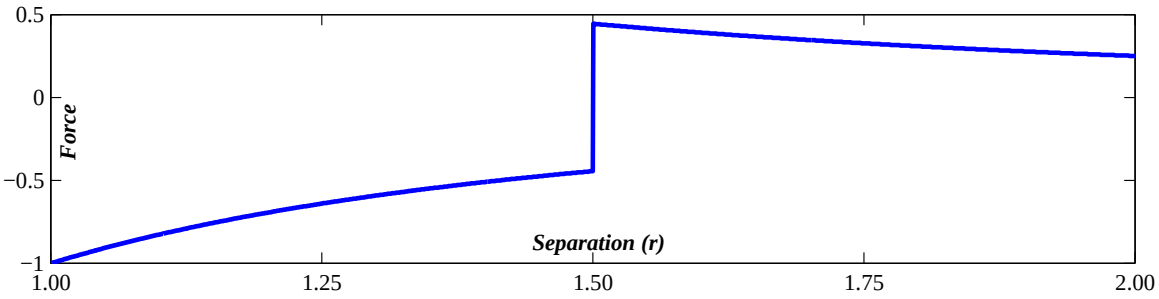


Figure 14.5: Plot of a sample force function used by the *AP* asset controller, with $r_{\text{desired}} = 1.5$ and $p = 2$.

- *Column:* is a variation on plain vertical movement of the UAVs, except the world is assumed to be toroidal, so that the assets "wrap around" in the same vertical column when moving past the top and bottom edges of the surveillance area. Modeling such behavior is helpful when performing statistical analysis of the surveillance problem.

- *LJ:* is another physics-based asset controller which uses a generalized Lennard-Jones force function:

$$\vec{F} = 24\varepsilon \left[\frac{d(\sigma_1 + \sigma_2)^6}{r^7} - \frac{2c(\sigma_1 + \sigma_2)^{12}}{r^{13}} \right]$$

  to facilitate UAV formation building (see Fig. 14.6). The two fractions in the above formula control the balance between the attractive and repulsive components of the force with respect to the current UAV separation radius r and the desired separation radii σ_1 and σ_2 for each pair of assets. As in the case of the AP controller, parameters may be set with the slider bars on the front panel, or can be loaded from a GA evolution trace file. Note that this controller uses the AP force law when computing the force between an asset and a surveillance area boundary.

- *LJMass:* is very similar to the LJ asset controller, but it attempts to account for the foliage growth (using the foliage sensor mounted on each UAV) by slowing down the UAV near the boundaries of the forest.

- *MagicCarpetArea:* is an asset controller useful in testing statistical models of the surveillance problem, in which assets "appear" at random locations throughout the world during the simulation. UAV locations are selected using a uniform random

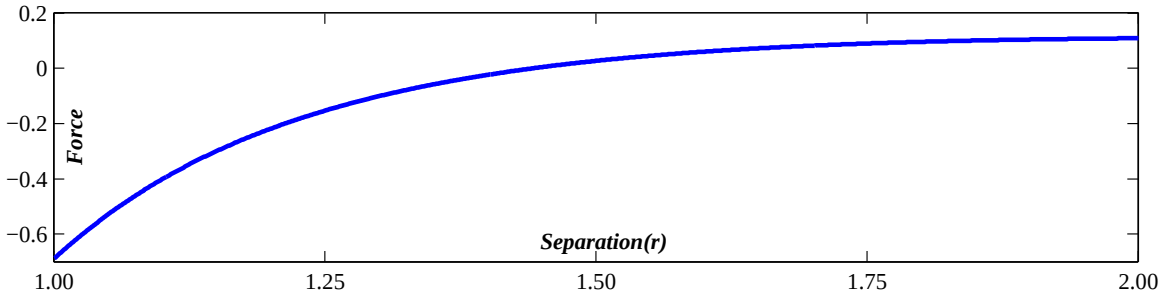


Figure 14.6: Plot of a sample force function used by the LJ asset controller, with $\sigma_1 = \sigma_2 = 1.5$.

distribution such that target sensor footprints do not overlap. This controller ignores the maximum asset speed specified in the `conf/asset.txt` file.

- *MagicCarpetColumn:* similar in spirit to the *MagicCarpetArea*, this controller causes assets to appear at random locations, but restricts the surveillance region to a single column, so that the x coordinate of the asset remains fixed, as the y coordinate changes randomly, without target sensor footprint overlap between adjacent asset locations.

- *SL:* a realistic asset controller which ignores foliage distribution, and simply moves an asset in a straight line at a constant speed until it reaches a world boundary, at which point the UAV will "reflect" from the edge by reversing its velocity component in the direction perpendicular to the boundary, similar to how a billiard ball bounces from an edge of the table.

- *SLAF:* is an asset controller which relies on a simple UAV foliage sensor, and is based on the premise that any surveillance time spent over foliage (which inhibits target detection) is in some sense "wasted", and thus attempts to minimize UAV flight time over forested areas. The controller assumes that the only output available from the tree sensor is the percentage of foliage cover in the sensor's view-field. When the sensed canopy cover exceeds a certain threshold (currently set to 50%), the UAV reverses its course, and begins moving in the opposite direction. Should the asset be initially deployed directly over the foliage, it will continue moving in its initial direction until it leaves the foliage, or an interaction with a world boundary causes it to turn around; thus given sufficient time, the asset will find a foliage-free region. In the absence of the foliage, this controller's behavior is equivalent to the *SL* controller (*SLAF* is an abbreviation of Straight Line Avoid Forest).

- *SSLAF:* a natural extension of the SLAF strategy, the Super SLAF controller makes use of a more sophisticated foliage sensor, which can compute the spatial distribution of the trees within the sensor's view. At each time step, a SSLAF-controlled UAV uses its forest sensor to compute the percentage of foliage visible through the sensor, and if that percentage exceeds a specified threshold (the default is 10%), the asset then estimates the mass distribution of the tree canopy underneath, and moves directly

away from the center of the foliage mass. Once the detected foliage percentage drops below the threshold, the asset will switch into its normal surveillance mode, continuing to move with its present velocity until an encounter with another patch of foliage or a bounce from the surveillance perimeter occurs.

- *Side:* controller produces linear "sweeping bands" of target sensor coverage by moving the UAVs back and forth along a linear path between two points on world edges. The location and direction of these surveillance corridors is determined by the initial location and velocity of each UAV, and may not include areas near the world boundary, as the assets may reverse their course right after detecting the edge of the surveillance area, and before scanning the boundary region with their target sensor.

- *Stationary:* assets hover over a fixed location in the surveillance perimeter, scanning any targets that may pass underneath. This is a simple controller that ignores the asset speed parameter in the `asset.txt` configuration file, and does not attempt to navigate around areas hidden by the foliage. Please keep in mind that the *Avoid* target strategy is 100% successful in avoiding detection by the *Stationary* assets.

As mentioned in the preceding description of the asset controllers, the maximum UAV speed is given by the `assetSpeed=`*float* parameter in the `conf/asset.txt` file, and it specifies the maximum distance (in terms of the world size units) that any given asset may travel in one simulation time step. The actual speed of an asset during the simulation might vary due to the UAV formation requirements or other constraints imposed by the specific controller. Also, because of the extensive use of the random data by some of the controllers, any particular sequence of random numbers and the corresponding change in behavior may be regenerated by using the same, non-negative value for the `assetSeed=`*integer* parameter; otherwise, a negative value for this parameter, which is the default, will cause the simulator to use the current system time in seconds, resulting in a new random asset initialization each time the simulator is run.

14.4 Genetic Algorithm Module

The SURVE toolkit includes a Genetic Algorithm (GA) module, whose purpose is to aid in the investigation of the impact of different force law options and parameter ranges on target detection rate. For the UAV asset controllers based on a generalized interaction potential, such as Artificial Physics (AP) or Lennard-Jones (LJ), it is sometimes possible to optimize the force function parameters in order to achieve improved surveillance performance. We accomplish this task by using genetic algorithms. Genetic algorithms can be thought of as an optimization technique, inspired by the process of the biological evolution. During execution, individual parameter sets in the population are mutated and recombined based on their performance (or fitness) in a given surveillance simulation run. The individuals with higher fitness than the average fitness of the population will survive through reproduction, and will contribute their genetic makeup to the future generations. One of the major reasons to employ a population-based stochastic algorithm is the rapid emergence of well-performing individuals, as well as the robustness of the individual performance improvements across a wide range of environmental configurations.

Typically, we use a population size of 100 individuals, and set the evolution termination criteria of the algorithm to 150 generations. The GA module generates an initial population of different potential function parameter sets using the random number seed specified in the `conf/gaapconf.txt` or `conf/galjconf.txt` files, so by changing this random seed, the program can generate many different sets of candidate parameter sets. Hence, every individual in the population represents one possible instantiation of either AP or LJ potential function optimization.

The following is a step-by-step guide on how to run the GA module, and how to set parameter settings of the potential functions using the configuration files, along with an explanation of the GA fitness function, data generation and storage, and the mechanism for visualizing the GA-evolved asset behavior.

14.4.1 Running the GA

The GA module included with the toolkit is located in the `ga` directory. In a UNIX-like operating environment, a user may simply execute the `runGA.sh` shell script. Otherwise, manually invoke the `java ga.PICKGA` class. Both of these options will begin evolution of parameters for the physics-based asset controller specified in the `conf/general.txt` file. If the `gavisual=`*true*|*false* parameter in the `conf/galjconf.txt` or `conf/gaapconf.txt` is set to *true*, then a window similar to the one shown in Fig. 14.1 (but without the information and control panels) will appear, showing asset behavior for the current individual parameter set. Even if the GA visualization window is initially enabled, closing it at a later time will not affect the GA run. If needed, pressing the `Ctrl+C` key combination (or the equivalent function of an Integrated Development Environment) should terminate the GA process.

14.4.2 Parameter Settings

The selection of the potential function for GA optimization is made via the `assetControl` parameter in the `conf/general.txt` file, where the three valid evolutionary asset types are: `assetcontrol.AP`, `assetcontrol.LJ`, and `assetcontrol.LJMass`. The LJMass controller, unlike the simpler LJ algorithm, attempts to minimize the amount of time the assets spend over the foliage-covered areas.

In addition to specifying the asset controller, do not forget to check the setting of the target controller, specified by the `targetControl` parameter. Since the GA will measure the reproductive fitness of individuals based on their target-detecting rate, the choice of a target movement strategy will have a strong influence on the type of the individuals discovered through evolution.

Force Functions' Parameter Ranges

There are two GA-specific configuration files in the `conf` folder. Evolution of the AP assets is controlled by the `conf/gaapconf.txt` file, while both the LJ and LJMass asset behaviors evolve according to the parameters in the `conf/galjconf.txt` file. Table 14.1 shows the range of valid values for AP populations, and Table 14.2 lists the valid ranges of the LJ and LJMass individuals.

	Ga.	$F_{max}a$.	Pa.	Gw.	$F_{max}w$.	Pw.	CoeffA	CoeffB	CoeffC	Fric
Min.	100	1	0.1	100	1	0.1	5	5	5	0
Max.	5000	5	2.0	5000	5	2.0	22	22	22	1

Table 14.1: Ranges of Values for AP Individuals

	ϵ	$F_{max}a$.	c	d	Gw.	$F_{max}w$.	Pw.	CoeffA	CoeffB	CoeffC	Fric
Min.	1	1	1	1	100	1	0.1	5	5	5	0
Max.	50	5	20	20	5000	5	2.0	22	22	22	1

Table 14.2: Ranges of Values for LJ and LJMass Individual

Explanation of the Parameters

The AP individuals:

- Ga. - gravitational force of asset-asset interactions.
- $F_{max}a$. - maximum force of asset-asset interactions.
- Pa. - exponent of the force law for asset-asset interactions.
- Gw. - gravitational force of wall-asset interactions.
- $F_{max}w$. - maximum force of wall-asset interactions.
- Pw. - exponent of the force law for wall-asset interactions.
- CoeffA - coefficient a of the second degree polynomial that determines the radius of an asset.
- CoeffB - coefficient b of the second degree polynomial that determines the radius of an asset.
- CoeffC - coefficient c of the second degree polynomial that determines the radius of an asset.
- Fric - friction in the system.

The LJ and LJMass individuals:

- ϵ - depth of the potential well.
- $\mathrm{F}_{\max}a.$ - maximum force of asset-asset interactions.
- c - non-negative attractive component.
- d - non-negative repulsive component.
- G$w.$ - gravitational force of wall-asset interactions.
- $\mathrm{F}_{\max}w.$ - maximum force of wall-asset interactions.
- P$w.$ - exponent of the force law for wall-asset interactions.
- CoeffA - coefficient a of the second degree polynomial that determines the radius of an asset.
- CoeffB - coefficient b of the second degree polynomial that determines the radius of an asset.
- CoeffC - coefficient c of the second degree polynomial that determines the radius of an asset.
- Fric - friction in the system.

Common Evolution Parameters

The other common parameters stored in both `gaapconf.txt` and `galjconf.txt` files are the population size (`populationsize=`*integer*), number of generations (`generations=`*integer*), number of parameters in each individual, i.e., the length of an individual (`length=`*integer*), mutation rate (`mutationrate=`*float*), crossover rate (`crossoverrate=`*float*), GA random seed (`gaseed=`*integer*), and evolution visualization preference (`gavisual=`*true*|*false*). All of these settings can be altered to fit a variety of experimental needs. Keep in mind that the negative value for the `gaseed` parameter will automatically set the pseudo-random number generator seed to the current system time, but otherwise will keep reusing the same random number sequence, resulting in the identical initial population for each new GA run.

```
Asset Controller Requested--> AP
...Initiate GA for Physicomimetics...
Log Level 0: Basic logging information
...Growing Forest
...Forest Growth Complete: 0.0
...% found 13 % inside 100.0
...Growing Forest
...Forest Growth Complete: 0.0
...% found 21 % inside 100.0
...Growing Forest
...Forest Growth Complete: 0.0
...% found 17 % inside 100.0
...Growing Forest
...Forest Growth Complete: 0.0
...% found 12 % inside 100.0
...Growing Forest
...Forest Growth Complete: 0.0
...% found 15 % inside 100.0
GA Individual Fitness: 15.6
```

Figure 14.7: Sample runtime output of the SURVE GA module.

14.4.3 Computing Fitness of Force Function Parameter Sets

In order to evolve the most fit individuals in the population, the simulator executes a surveillance scenario with a randomly generated forest configuration, and then calculates the fitness for each individual force law parameter set in the population. The fitness, or the performance, of an individual is the percentage of successful target localizations by the entire agent collective during the surveillance activity, whose duration is set by the `timeSteps` parameter in the `conf/general.txt` configuration file. The following formula is used when computing the fitness of an individual:

$$\text{fitness} = \frac{\text{targets seen}}{\text{total number of visible targets}} * 100$$

In addition, the fitness of each individual is also averaged over a specified number of different random forest configurations (currently fixed at 5), where the initial asset and target locations are different for each new forest instantiation. A sample of the runtime GA module output is shown in Fig. 14.7.

The selection of the individuals for reproduction is performed using the Baker's Stochastic Universal Sampling algorithm, and the offspring is generated using a one-point crossover, with the rate specified by the `crossoverrate` configuration parameter, with some additional individual changes possibly introduced via a Gaussian mutation, governed by the `mutationrate` parameter. The Gaussian random mutation used by the SURVE toolkit comes from a standard normal distribution with a 0 mean and a variance of 1. The implementation of the Gaussian mutation conforms closely to the widely accepted methodology in the evolutionary algorithms literature. In this case, the Gaussian random mutation method has a 1/(individual length) probability of mutation. Note that this implementation will reject any offspring whose parameter values fall outside of the accepted range (see Tables 14.1 and 14.2).

14.4.4 Data Generation and Storage

During execution, the GA module generates three text files which contain performance-related information about the population of the force function parameters. These files are stored in the `log` folder, inside the `surveillance2` directory. Table 14.3 lists the names along with a brief description of the three output files generated by the evolution module.

Each output file contains the sequential generation number, as well as the sequential identification of each individual in the population along with that individual's fitness (i.e., surveillance performance). The `best_worst` file only shows the best and the worst performing individuals for each generation, and is helpful in evaluating the efficacy of the evolutionary strategy. The history file contains the generation count, an individual's location in the population, and an individual's random seed, but does not show the fitness of the individuals. Each output file lists all of the force function parameter values represented by each GA individual.

Contents	*AP*	*LJ*	*LJMass*
All Individuals	`GAAPresults`	`GALJresults`	`GALJMresults`
Best and Worst Individuals	`APbest_worst`	`LJbest_worst`	`LJMbest_worst`
All Individuals with Random Seeds	`APhistory`	`LJhistory`	`LJMhistory`

Table 14.3: Output files (*.txt) from the GA module.

14.4.5 Using GA Individuals

The SURVE toolkit provides the functionality to directly process the output files generated by the GA module by first loading the entire history file, and then instantiating the asset controller with the parameters from the evolved individual. Once the GA completes its run and generates the output files, the GA menu can be used to process that output and to conduct experiments with the evolved individuals (for more information on the GA menu, please see Section 14.6 on simulator menus). The GA menu contains two actions: Load GA History and Load GA Individual. To create an asset controller based on a GA individual, first load the GA history file, and then load a specific GA individual by specifying both the generation and individual identification tags. The Load GA History menu item will display a "File Open" dialog window initially positioned in the `log` directory, through which the user should select the correct GA history file and then press the Open button to load the history file into the SURVE application. Then choose the Load GA Individual menu item, which will display a window similar to the one shown in Fig. 14.8, where the generation and individual identification numbers can be specified. Once this information is entered, the user can observe this individual's surveillance behavior by simply clicking the Start button on the main control panel of the simulator (see Fig. 14.1).

Figure 14.8: Load GA Individual window.

14.5 Simulator Parameters

Upon startup, SURVE reads several text-based configuration files from the `conf` directory to initialize its components. The list below contains the names of all of the files, a brief description of the simulator functionality that depends on that file, and then provides a detailed

explanation of each configuration parameter, its default value, and in parenthesis, the type of the parameter. One important fact to remember is that configuration parameter values are only examined during SURVE initialization, and changing the parameter values in the configuration files while the simulator is running will have no effect on the currently running instance of the simulator. To change system parameters at run time, please use the control panel sliders and option menus in the SURVE graphical user interface (see Section 14.6).

14.5.1 General Parameters

The `conf/general.txt` configuration file provides values for target control parameters, as well as the general SURVE simulator parameters.

- `agents=5` (*positive integer*) - number of UAV assets in the simulation.
- `targets=100` (*non-negative integer*) - number of targets in the simulation.
- `targetSeed=-1` (*integer*) - target random number generator seed; system time is used when the value is negative.
- `targetSpeed=0.4` (*non-negative float*) - maximum speed of the targets.
- `targetOffset=-1.0` (*float*) - target initial position horizontal offset in terms of the world size.
- `targetControl=targetcontrol.Sneaky` (*string*) - target movement strategy, entered as a fully-qualified Java class name.
- `assetControl=assetcontrol.SL` (*string*) - asset control strategy, specified as a fully-qualified Java class name.
- `size=200` (*positive integer*) - world size (in screen pixels).
- `timesteps=1000` (*positive integer*) - number of simulation steps in each surveillance scenario.
- `sleeptime=0` (*non-negative integer*) - number of milliseconds between screen updates in visual experiments.

- `debug=0,3` (*integer list*) - selection of debugging events to be reported to the console. Currently, four types of debugging events are available: (0) basic logging information, (1) random seed information, (2) step-by-step target localization information, (3) target collision errors.

Please keep in mind that both asset and target controller strategies must be named in the configuration file with their fully-qualified Java class names (i.e., the name of the strategy must be prepended with the `assetcontrol` or `targetcontrol` Java package name).

14.5.2 Asset Parameters

The controlling parameters for the UAV assets are stored in the `conf/asset.txt` file, and are enumerated below:

- `assetSpeed=3.0` (*positive float*) - maximum speed of an asset.
- `targetRadius=5` (*positive integer*) - radius of the ground-level target sensor footprint.
- `targetSensorType=circle` (*string*) - shape of the ground-level target sensor footprint; possible values are *circle* and *square*.
- `targetSensorIncludeBounds=0` (*integers 0 or 1*) - Boolean flag which controls the distance comparison operator used during target detection; when the parameter value is 0, operator $<$ is used, otherwise $\leq$ comparison is employed.
- `foliageRadius=20` (*positive integer*) - radius of the ground-level foliage sensor footprint (always treated as a circle).
- `detectionProb=1.0` (*float between 0 and 1*) - probability of successfully detecting a target when it is within the sensor range.
- `assetSeed=-1` (*integer*) - random number generator seed for stochastic components of asset behavior; if the value is negative, then the current system time in seconds will be used.

Initial asset positions are selected randomly from either a uniform or a normal distribution, depending on the asset controller. Most of the time, an asset will move with the speed given

by the value of the `assetSpeed` parameter, unless interactions with the neighboring assets cause the UAV to slow down.

14.5.3 Forest Parameters

The foliage production procedure uses the values of the following parameters, which are located in the `conf/forest.txt` configuration file:

- `growthStrategy=env.NaturalPineCones` (*string*) - fully-qualified Java class name for the tree placement strategy; currently, the only choice is `env.NaturalPineCones`.
- `seedRadius=8` (*positive integer*) - maximum distance that a seed can fall away from its parent tree.
- `seedStrength=0.2` (*float between 0 and 1*) - probability that a seed will grow into a tree (i.e., seed viability).
- `seedDecay=0.9` (*positive float*) - seed viability decay rate.
- `forestSize=10` (*positive integer*) - initial size of the forest (i.e., the number of starter trees, randomly distributed over the surveillance area).
- `forestCoverage=0.2` (*float between 0 and 1*) - desired amount of foliage, expressed as the fraction of the total area of the simulated world.
- `forestSeed=-1` (*integer*) - random number generator seed for the forest algorithm; system time will be used as the seed when this value is negative.

The first step of the forest generation process selects `forestSize` initial locations randomly, in order to plant the first parent trees. During the next step, the parent trees will produce seeds within the circle of `seedRadius`, and each seed will have a `seedStrength` probability of growing into a new tree. Note that if multiple nearby parent trees deposit seeds in the same location, then the probability that a new tree will appear at that location increases in direct proportion to the number of seeds planted there. Since not all of the planted seeds will become trees during the first iteration, the seeds that failed to produce a new tree will lose some of their vitality, with the rate of seed viability decline given by the value of the

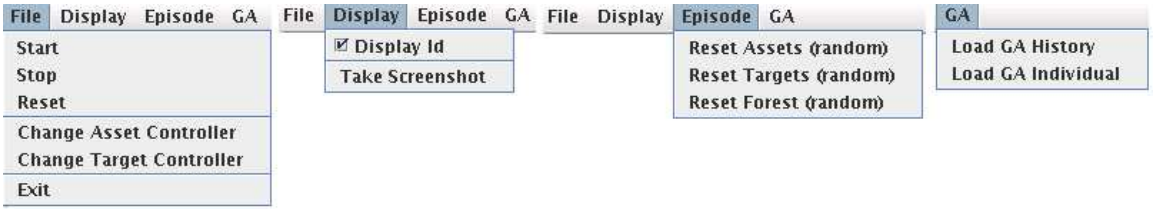


Figure 14.9: Expanded view of the SURVE application menubar.

`seedDecay` parameter. The seeding algorithm continues until at least `forestCoverage` of the entire surveillance area is occupied by trees, thus it may be possible for the final foliage cover to exceed the specified threshold by a small amount.

14.6 User Interface

This section provides an overview of the SURVE graphical user interface (GUI). Whenever possible, the design strives for simplicity and intuitive user experience, so many of the options and buttons should be self-explanatory. Because of the desire to keep the interface simple, only a subset of the simulator parameters can be changed via the GUI; the rest of the configuration should be performed through the configuration file interface (see Section 14.5 for more information).

14.6.1 Menubar

The main application menubar provides the means to quickly and easily change the most important simulation parameters, and to perform large-scale changes to the surveillance experiments, such as altering the asset control strategy, or building a new foliage map. A composite image of the expanded view of the main SURVE menus is shown in Fig. 14.9. The simulator can also be operated via the the button and slider control panel, shown in Fig. 14.13.

File Menu

The File menu contains frequently used simulation commands: start and stop will initiate or terminate the simulation, the reset will terminate the current simulation and reposition the

assets and the targets using the current system time as the random number generator seed. The same functionality is also available through the three buttons in the control panel (see Fig. 14.13). The menu items for changing asset and target controller strategies will bring up a new window where the new controller type can be selected (see Fig. 14.10). All of the default asset and target controllers that are distributed with the SURVE simulator can be activated by using these menus. Selection of a new controller will cause the corresponding random number generator to be initialized with the current system time, and the position of the agents will be reset. As the name suggests, the Exit menu item will terminate the simulation and quit the SURVE application.

Display Menu

The Display menu allows the user to toggle the on-screen marking of each asset with its identification number, which is extremely helpful when tracking the trajectory of a specific asset. The Take Screenshot item creates a JPEG image of the current SURVE state, pausing the simulation if necessary. Only the surveillance area, including the forest, assets, and targets is saved in the image; the GUI elements and simulator controls are not captured in the image (to obtain a complete screenshot of the entire SURVE window use a capture utility provided by the operating system). Clicking the Take Screenshot button will open a "Save File" dialog box (see Fig. 14.11), where the file name for the new image should be

Figure 14.10: Selecting a new asset and target controller.

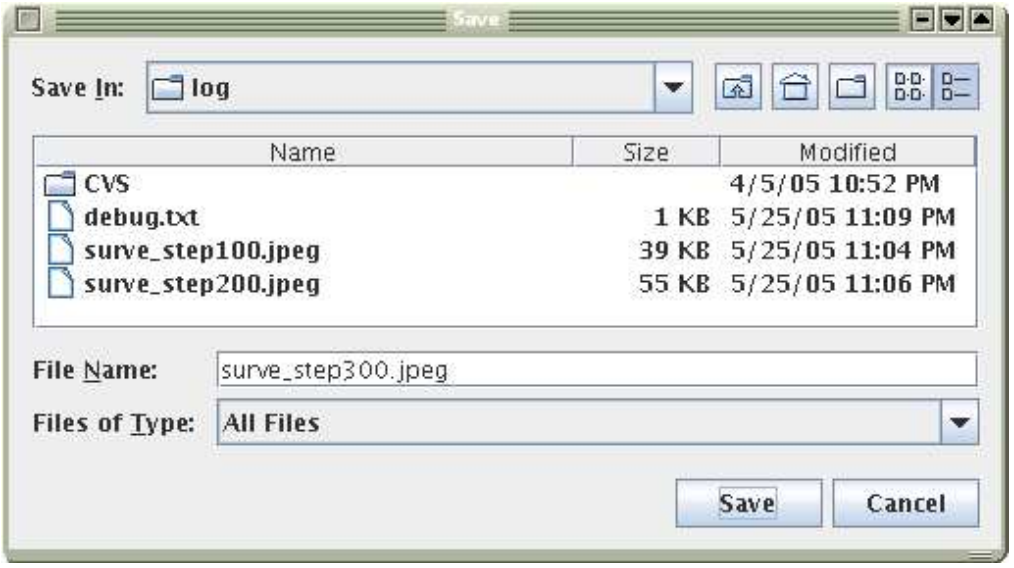


Figure 14.11: Saving a screenshot of the SURVE simulation.

specified.

Episode Menu

The Episode menu provides a quick way to regenerate a forest, reposition targets, and to re-initialize the UAV assets. The first menu action Reset Assets will set the new asset positions using the current system time as the seed for the random number generator. Executing this action will relocate the assets, but will leave the targets in their current positions. However, the count of targets detected by the assets will be set to zero, since resetting the assets creates a completely new set of surveillance agents. Use the Reset Targets menu item for restarting the targets, which again will use the current system time as the random seed, and will place the new targets using this new random number sequence. The old UAV asset locations will not be affected, but since a completely new target group is created as the result of executing this action, the count of targets previously detected will again be set to zero. The last function in this menu, Reset Forest, will change the forest map by obtaining the new random number generator seed from the current system time, and will regrow the forest with the same area coverage as before, but now with different locations of the starter trees. While both the assets and the targets will keep their current

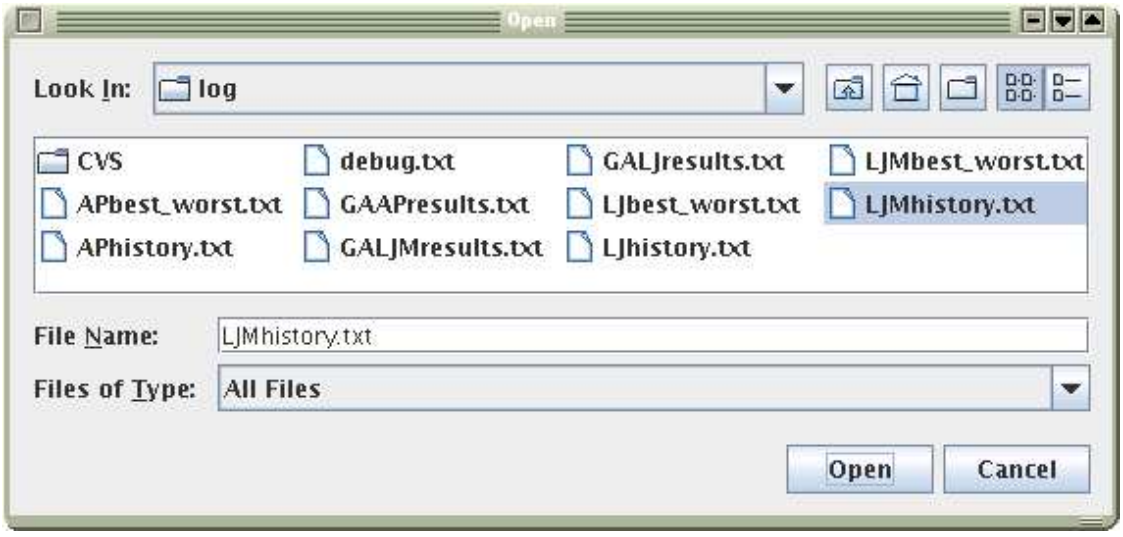


Figure 14.12: Loading GA evolution history file.

positions after the change in the foliage map, all of the counters (including detected targets and simulation time) will be reset to zero.

GA Menu

The GA menu can only be used when the GA module execution history output file is available (see Section 14.4 for a detailed explanation of the SURVE evolutionary algorithm feature). The purpose of this menu is to allow for a run-time visualization of the asset surveillance behavior based on the force function parameters obtained with the genetic algorithm. In order to visualize a given parameter set, you need to know the generation and the individual identification number of the GA individual you wish to instantiate. Once these are known, choose the Load GA History menu item, which will display the "Open File" window similar to the one shown in Fig. 14.12, where you should select the correct history file, and then click the Open button to load all of the parameter sets into the application. Then, activate the GA menu again, and this time select the Load GA Individual option, which will display the entry dialog (see Fig. 14.8) where you will input the generation and individual ID tags for the specific parameter set you wish to load. When you click the OK button, the asset controller parameters will be extracted from the GA history file and activated, which will reset the asset positions and update the asset control slider panel. *Note:* before loading

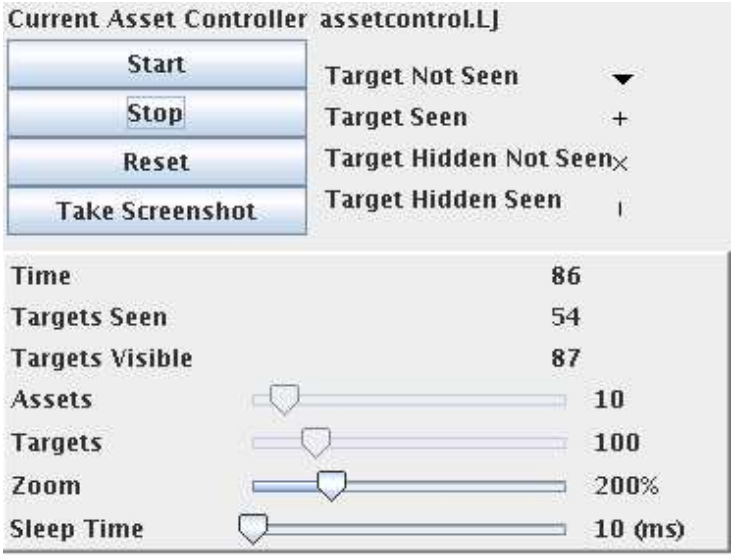


Figure 14.13: SURVE control panel.

the GA history output file, you must make sure that the current asset controller is of the same type as the asset controller used to generate the GA history file; in other words, if you load the `LJhistory.txt` file, then make sure that the current asset controller is set to `assetcontrol.LJ`. For greater convenience, you may select the Load GA Individual option multiple times with different generation and individual ID tags without restarting the simulator, but keep in mind that the current target locations and the forest map are preserved when you load a new set of asset control parameters. If you wish to change the target or foliage placement, please use the appropriate item from the Episode menu. Also, if you mistakingly activate the Load GA Individual item before loading a history file, then SURVE will first prompt you for the history file, and then will allow you to select an individual from that file.

14.6.2 Simulation Control Panel

The main SURVE control panel is shown in Fig. 14.13. The four buttons on the left hand side of the panel are available for user convenience, and are identical in functionality to the corresponding File and Display menu items. The panel also contains an information area where the current asset controller and the target display legend are shown. Note that the

target display symbol legend, located to the right of the simulation control buttons, is useful in identifying the four different surveillance classifications that each target may have, which are:

1. visible, but not detected (downward triangle symbol)
2. detected (plus sign symbol)
3. undetected target obscured by foliage (multiplication symbol)
4. detected target obscured by foliage (vertical pipe symbol)

Below the legend, you will find the information about the current simulation, which includes simulation time step, the number of detected targets, and the number of targets that have been visible at least once up to this time step. For stationary targets, the visible counter will not change as the simulation runs, but for moving targets this number will normally increase as the simulation progresses, since more and more targets will be moving through the open surveillance areas.

Underneath the simulator statistics are two disabled sliders for the number of assets and targets. These numbers cannot be changed while the simulator is running, and should only be set in the `conf/general.txt` file as the values of the `agents` and `targets` parameters. The zoom control bar allows for magnification of the surveillance display region, however, the display region will always use as much screen space as possible, thus it may be necessary to resize the SURVE window in order to shrink the display in some cases. The control panel also includes the slider for adjusting the sleep time, which affects the delay between the consecutive screen updates, and increasing this delay will be helpful when tracing the trajectory of individual assets or targets.

14.6.3 Asset Controller Panel

When you replace the asset controller strategy via the File menu, you will notice that the slider control panel in the lower right of the SURVE application window changes (see Fig. 14.14). This happens because each controller has a different set of parameters that may be modified during the run of the simulation, and some controllers may have no parameters, resulting in an empty GUI panel.

Asset controllers based on physical interactions between neighboring assets and the environment provide the most flexible and powerful control interface, through which the user can alter the strength and type of asset-to-asset and asset-to-environment forces. The two parameter sliders available in the control panel, which do not appear in the canonical AP and LJ force equations, are the $F_{\max}$ and the friction controls. Parameter $F_{\max}$ limits the force resulting from assets moving too close to each other or the surveillance perimeter. The friction parameter determines how much of the assets' kinetic energy is dissipated during one simulation time step.

14.7 Batch Mode Experiments

One extremely powerful feature of the SURVE toolkit is the ability to run a large number of surveillance experiments in the non-interactive batch mode. Instead of displaying the graphic user interface and rendering every detail for each simulation time step, the batch mode simply executes the simulation with visualization disabled, and writes the performance statistics to an output file for later processing and analysis. Because this mode of surveillance experimentation requires a great deal of customization of system parameters, the default SURVE distribution includes a sample batch mode class file called `IEEETester` which illustrates how you can create new experiments to better fit your investigation.

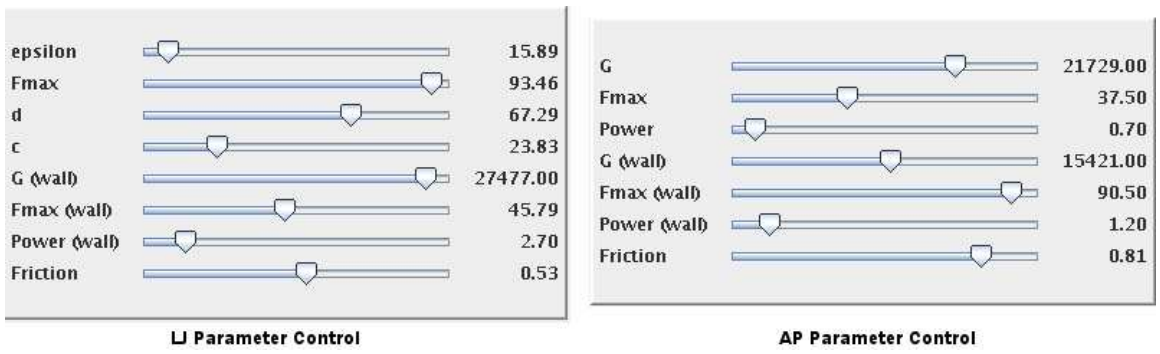


Figure 14.14: SURVE asset controller adjustment panels.

14.7.1 IEEE Tester (batch mode)

The `IEEETester` class file will execute a number of SURVE simulations for a given foliage cover, number of UAV assets, and target detection probability, overriding the corresponding parameters from the configuration files. Its purpose is to evaluate the robustness of a surveillance strategy by repeatedly executing it for a combination of different foliage maps and initial target positions. The format of the text output file follows the MATLAB M-script conventions, with the surveillance simulation results stored as 3 row vectors of floating point numbers, which correspond to the percentage of visible targets found by SL, SLAF, and LJ asset controller strategies. You can change the number of foliage maps generated in each run by modifying the initialization value of the `forestRuns` variable in the `experiments/IEEETester.java` source file. Likewise, the number of different target starting positions (per forest map) is controlled via the `testRuns` variable in that source file. After making changes to the source code, remember to compile the file with `javac experiments/IEEETester.java`. If working on a UNIX-like operating system, the `Makefile` included in the SURVE source distribution can be used to quickly rebuild the application.